Energy efficient path planning:

The effectiveness of a Q-learning algorithm

in saving energy

Prepared By:

Samuel Ogunniyi

Department of Electrical Engineering

University of Cape Town

Supervisor:

Mr. Mohohlo Tsoeu

This dissertation is submitted in fulfilment of the academic requirements for the

Masters of Science Degree in Electrical Engineering, Msc (Eng)

at the University of Cape Town

November 2014

I. PLAGARISM DECLARATION

1. I know that plagiarism is wrong. Plagiarism is using another's work and to pretend that it is ones own.

2. I have used the Institute of electrical and electronics engineers (IEEE) convention as the convention for citation and referencing. Each significant contribution to, and quotation in, this essay/report/project/... from the work, or works of other people has been attributed and has cited and referenced.

3. This essay/report/project... is my own work.

4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as his or her own work.

5. I acknowledge that copying someone else's assignment or essay, or part of it, is wrong, and declare that this is my own work

SIGNATURE: Signed by candidate

DATE: 18/11/2014

II. ACKNOWLEDGEMENTS

First and foremost I would like to thank God for giving me the strength to undertake this study in spite of all the challenges that I have encountered.

To my supervisor, Mr Tsoeu I say a big thank you for guiding me along the way. Your assistance when needed has helped to bring this study to a logical conclusion. Likewise, your advice have helped to overcome the various obstacles that confronted me in the pursuance of this study am extremely grateful to you and I appreciate everything you did to enhance and enlarge my intellectual horizon.

Then thank you classmates, colleagues and friends for the assistance they have provided me during the last two years

Last but not the least I want to give a special thank you to my parents who have nurtured, encouraged, and supported me spiritually, morally and financially over the years and for being there for me despite the trials we have faced as a family in the past three years.

I dedicate this thesis to the sweet memory my late Older Brother and hero, Mr David Ogunniyi who left us suddenly to a better place at a very young age

III. ABSTRACT

In this thesis the author investigated the use of a Q-learning based path planning algorithm to investigate how effective it is in saving energy. It is important to pursue any means to save energy in this day and age, due to the excessive exploitation of natural resources and in order to prevent drops in production in industrial environments where less downtime is necessary or other applications where a mobile robot running out of energy can be costly or even disastrous, such as search and rescue operations or dangerous environment navigation. The study was undertaken by implementing a Q-learning based path planning algorithm in several unstructured and unknown environments. A cell decomposition method was used to generate the search space representation of the environments, within which the algorithm operated. The results show that the Q-learning path planner paths on average consumed 3.04% less energy than the A* path planning algorithm, in a square 20% obstacle density environment. The Q-learning path planner consumed on average 5.79% more energy than the least energy paths for the same environment. In the case of rectangular environments, the Q-learning path planning algorithm uses 1.68% less energy, than the A* path algorithm and 3.26 % more energy than the least energy paths. The implication of this study is to highlight the need for the use of learning algorithm in attempting to solve problems whose existing solutions are not learning based, in order to obtain better solutions.

IV. ABBRIEVIATIONS

C-SPACE	Configuration space
BFS	Breadth first search
DFS	Depth first search
MDP	Markov Decision Process

V. CONTENTS

VI. LIST OF FIGURES

VII. LIST OF TABLES

1. INTRODUCTION

1.1 Rationale

Most robots in use in industrial environments today are stationary and huge. They are typically used in factories for repetitive tasks such as assembly operations, welding, dispensing and processing. These robots can perform tasks that humans can do with more precision. Since they do not tire like humans, they are well suited to the continuous and repetitive nature of such tasks, while maintaining precision [1] [2]. However most industrial robots are fixed and are limited in application flexibility due to lack of *mobility*. Mobile robots extend the capabilities of traditional fixed robots, possessing additional flexibility making new applications possible. These applications include: medical, surgical applications, dangerous situation navigation, ocean and even space exploration amongst numerous others [3].

Due to the demands placed on contemporary mobile robot units, they are generally seen to have limited use, unless they possess some form of autonomy [4]. Varying levels of autonomy from limited to fully autonomous are the basis for the concept, which the author refers to as the *intelligence of a mobile robot.* This can be explained simply as the ability of a mobile robot to learn how to perform particular tasks with limited supervision. A very popular problem in modern robotics literature is that of path planning.

The context the author wants to convey is not path planning in terms of manual programming of motions and paths, but rather through the learning of paths along which the mobile robots can navigate, through the use of path planning or search algorithms. The author intends to compare well-known learning algorithms to see what effect their output planned paths have on the energy consumption of a mobile robot. This will be elaborated upon in the following sections and chapters

1.2 Problem Statement

Generally path planning [5] is about finding a path between two points or nodes, and more specifically finding the optimal shortest path between the two points. For the context of this research this could be seen as, for example a path determined by a robot in order to navigate from a start location to a destination location in an unknown and unstructured environment. “Unknown” in this context means that the locations of the obstacles in the environment are not identified before the algorithm is run. The term “unstructured” in this context means that the positions of the obstacles are randomly generated and are not prearranged beforehand. A unique problem area arises from the computational complexities involved in a robot’s agent trying to navigate these types of environments. This prompts the author to investigate the use of a suitable algorithm, to attempt to find a path which allows a mobile robot to expend the least energy while navigating in the above mentioned unknown and unstructured environments. Taking into account that the geographic location of the target is known relative to the robot, the term unknown is defined here in terms of lack of information about the geographic positions of the obstacles in the environment.

1.3 Purpose of study

Most path planning algorithms operate in configuration spaces which are representations of the environment after a graph construction method such as the Voronoi diagrams method, occupancy grids method and vertex graph method, to name a few have been performed on said environments [6]. The purpose of this study was to evaluate the effectiveness of a Q-learning algorithm in producing an energy efficient path within occupancy grid decomposition. Another graph search algorithm, known as the A star algorithm, was used as an exemplar to demonstrate such effectiveness. The next section presents a brief theoretical framework. In the pursuance of the aim above, the author sought answers to the following questions:

1) How effective is the Q-learning algorithm proposed for the study, in generating planning energy saving paths?
2) How does the proposed Q-learning compared to the A* path planning algorithm?

1.4 Theoretical Framework

The theoretical framework underpinning this study is that of cell decomposition. Cell decomposition uses geometrical and algebraic techniques to transform the environmental information into a representation which can be understood by a robot, but is also extremely intuitive to human beings. It is a very popular approach to the path planning problem [7].

The basic principle behind the cell decomposition method is that, a graphical representation of an environment called the Configuration Space (C-SPACE) is divided into smaller polygon regions called cells. Each cell then can be classified as either non-traversable or free space with certain costs for traversing through them. The C-SPACE can be divided using different methods i.e. exact decomposition, approximate decomposition or fixed decomposition also known as an occupancy grid, which is used in this study [8].

A graph searching algorithm can then be used to find a path within the free space of the occupancy grid. In this study, the Q-learning algorithm is used as the graph searching algorithm as it is currently a well-liked and thoroughly researched algorithm in contemporary research endeavors [7] [9]. The Q-learning algorithm is a model free reinforcement learning algorithm developed by C. Watkins [10], and inspired by Sutton's 1984 method of temporal differences. It is useful in this study because it easily extends to path planning applications [10].

1.5 Limitations of the study

The following factors placed some constraints on the study

- Software available:

  The author was only able to use a student version of Matlab as that was what was cost effective to use.
- Hardware availability placed some constraints on what could be done.

  The power of the processor limited the number simulations run and the size of the maps used in the study were generally smaller than one would have liked.
- Convergence time of algorithms:

  The Q-learning algorithm is quite slow to convergence in large exploration environments.
- Complexity of other algorithms to be implemented:

  The complexity of the algorithm and the time available made it difficult to implement the other algorithm with which to compare the Q-learning path planner.

1.6 Delimitations

The delimitations of a study deal with the focus or scope of the study. The focus of this study is to measure the energy consumption of a mobile robot navigating a path generated by the Q-learning based path planner. More specifically, it is concerned with checking how effective Q-learning is at energy saving. A related aim is to compare the Q-learning algorithm with the A* algorithm as an exemplar mechanism for energy saving effectiveness.

1.7 Significance of the study

There are three general research areas in robot navigation: localization, motion control and path planning [6]. The area of path planning is of interest in this study. In the area of path planning researchers such as Donald, Brooks and Kambhamti *et al.* have endeavoured to obtain optimal paths in optimal times by manipulating how the workspace environments are represented [6]. This means that time was taken to ensure obstacles and open spaces were adequately represented, as well as choosing befitting algorithms which were robust enough to exploit data structures, all to facilitate better navigation.

Path planning is proposed to be used as a basis to facilitate the development of useful tools or methods for promoting reduced energy consumption and can be used as a learning tool for manufacturing industries to improve productivity. It can also be used by robot manufactures or research hubs, to have their robots operating more efficiently. This can possibly filter through to the secondary sector industries forming a better relationship between robotics development and economic growth, such as currently in several Asian countries. This is hopefully a small contribution to the research of reinforcement learning applications.

This study's goal is to help industries to improve their energy footprint through smarter and more versatile mobile robots. This goal can be achieved through equipping mobile robots to learn how to adjust their behaviours when encountering unknown environments. The output of this study is a dissertation paper from which researchers and other stakeholders can assimilate for further research endeavours or implementations.

1.8 Methodology

In order to achieve the goal of this thesis the author developed a Q-learning path planning simulator. Using this simulator the author was able to collect essential data through experiments developed to be used with the simulator. The first experiments tested that the Q-learning path planning algorithm works like it is supposed to i.e. that the algorithm can actually learn. The plots that show that the algorithm learns are included in Appendix A1.

The second set of experiments tested that the algorithm converges. In order to do this it is essential to obtain results of average number of steps and energy used through a range of iteration values. These results will be displayed as tables and boxplots and will be shown in section 7.1. Included in the second set of experiment was the test that the algorithm allows exploration i.e. not deterministic. This test was already completed as the algorithm relies on an e-greedy action select policy which introduces a certain amount of randomness to the action selection policy during navigation.

The third set of experiments tested the performance of the Q-learning algorithm against the A* algorithm, in terms of number of steps and energy consumption. The results are shown in section 7.4.

1.9 Operational definitions

Below are operational definitions for the context of this thesis

1. **Effectiveness**- natural inherent ability to achieve a result i.e. energy saving
2. **Autonomous** - self-governing [11]
3. **Reinforcement**- Any kind of stimulus which strengthens or increases the probability of a specific response [11] [12]
4. **Markov chain** - a model for a random process that evolves over time such that the states occupied in the future are independent of the states in the past given the current state [13]
5. **Markov decision problem (MDP)** - a model for a controlled random process in which an agent's choice of action determines the optimal policy for transitions of a Markov chain and leads to rewards (or costs) that need to be maximized (or minimized) [13] [14]
6. **Agent -** an agent is "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators" [13] [15].
7. **Policy** - a deterministic or stochastic scheme for choosing an action at every state or location more specifically in the context of the thesis is a "mapping from each state, $s_t \in S$, and action, $\alpha_t \in A$, to the probability $\pi(s, a)$ of taking action a when in state s". [13] [14].
8. **Reward** - an immediate, possibly stochastic, payoff that results from performing an action in a state [13].
9. **Value function** - a function defined over states, which gives an estimate of the total reward expected in the future, starting from each state, and following a particular policy [13].
10. **Discounting**- if rewards received in the far future are worth less than rewards received sooner; they are described as being discounted [13].
11. **Dynamic programming -** a collection of calculation techniques for finding a policy that maximizes reward or minimizes costs [13].
12. **Localization -** a determination of the place where something is [16].
13. **Holonomic -** If the controllable degree of freedom of a robot is equal to total degrees of freedom [86].
14. **Optimality** - The measure of how good a solution is with respect to the path cost function [87].

1.10 Assumptions

In order to achieve the objectives of this study some assumptions should be borne in mind such as:

- Initially the algorithm will be setup using a graphical representation of the environment
- In this study, the representation to be used is that of a small circular robot on configuration space decomposed by the occupancy grid method [8]. The result of the decomposition method is a connectivity graph along which paths can be planned along available nodes from start node to goal node.
- This means that any data about the dimensions of environment i.e. open spaces and obstacles can be assumed to have been obtained from cameras above the environment, image processed and converted into a 2 dimensional state representation of said environment for possible real-time implementation.
- The output motions of the path planner to the robot will be relative directions i.e. north, South, East, West, North-East, North-West, South-East and South West, assuming North is according to convention.
- The energy efficient calculations will be composed of the energy consumed during linear motion as well as the energies consumed during acceleration and deceleration of the robot when turning.

1.11 Thesis outline

Chapter 1 serves as an introduction to the study and establishes the theoretical framework underpinning the study. Chapter 2 introduces relevant path planning literature. Q-learning, which is the algorithm of interest in this study is introduced and explored in much detail in chapter 3. Chapter 4 discusses briefly literature pertaining to energy consumption practices for mobile robots. Chapter 5 presents some related work. Chapter 6 shows the methodology used to implement and evaluate the algorithms. Chapter 7 reports the findings and discusses them using relevant literature. Lastly Chapter 8 provides some conclusions and recommendations to wrap up the study.

2 PATH PLANNING:

Path planning research has been undertaken for decades. The interest in path planning rose due to the development of industrial manipulator robots. Manipulator robots could have many degrees of freedom and the path planning problem of having to move arms or limbs with accuracy was quite complex in relation to differential drive robots on flat surfaces. However the techniques developed for manipulator robots have been the stimulus for many of path planning techniques that were developed from mobile robots. The results have been that the previously developed path planning algorithms were simplified, as the degrees of freedom were greatly reduced and also less complicated, because most mobile robots operated at speeds which were low enough to almost completely ignore the dynamics and kinematics used by the industrial robots, for which the algorithms were originally developed [8].

2.1 Configuration Space

The representation within which path planning is done for both manipulator and mobile robots is called the configuration space or C-SPACE. It is defined as a set of parameters that completely specify the position and pose of an object, for e.g. the positions of a robot arm [17].This can be represented as a set of coordinates or points of k real values: $\boldsymbol{q_1}, \ldots, \boldsymbol{q_k}$, in a *k*-dimensional space of *k* degrees of freedom of an object. This means a complex 3D robot structure could be represented by a single *k*-dimensional point [8].

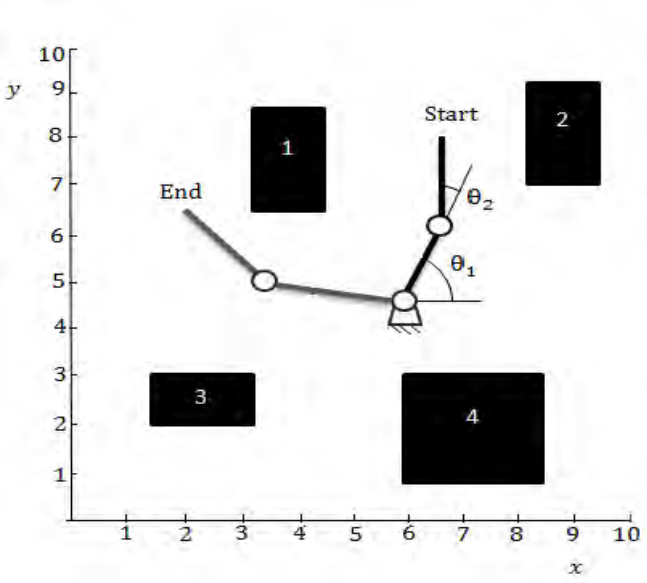

Figure 2.1: This schematic illustrates a physical environment containing a two-link planar robot's starting and ending positions. The obstacles in the environment are represented by the dark blocks numbered 1, 2, 3 and 4 (Adapted from [8]).

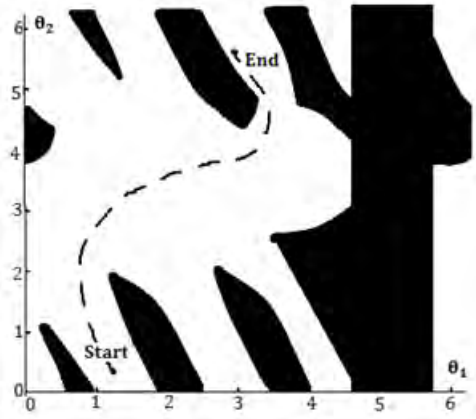

Figure 2.2: This figure shows the corresponding configuration space to figure 2.2 which includes the joints coordinates ($\boldsymbol{\theta_1}, \boldsymbol{\theta_2}$) and the path the joints have to take to reach the end position (Adapted from [8]).

2.2 Graph Construction Methods

There are three main types of environment representations with which path planning algorithms can be used, namely continuous geometric maps, decomposition based geometric maps and topological maps. Before any path planning scheme can be set up the actual continuous environment model must be converted into a precise discrete map suitable for a certain path planning algorithm. There are methods which have been identified to transform/construct the environmental map into a connectivity graph, which can be searched using path planning algorithms [8]. These methods will be discussed in the next subsections.

2.2.1 Visibility graph method

In the visibility graph method, any unobstructed vertices can be connected together using straight line segments. In this method, all the vertices/nodes at edges of geometrical shapes or obstacles can be connected to each by line segments. Any vertices that are unobstructed from the start and goal points are then connected by line segments. A path planner is then supposed to find a shortest path from the initial position to the goal along line segments on the visibility graph, bending only at the vertices of obstacles [8] [18].

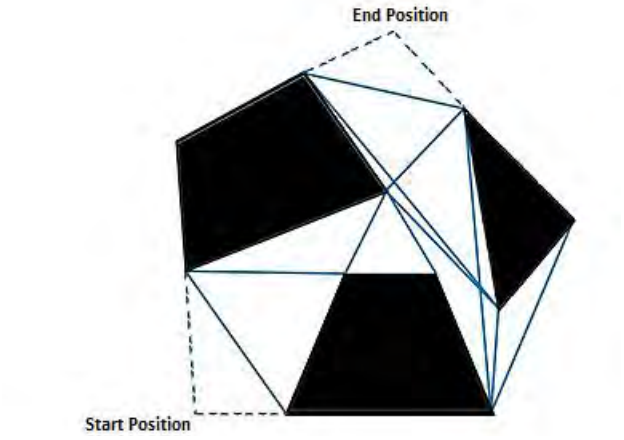

Figure 2.3: This figure shows a visibility graph. The nodes of a visibility graph are the start point, the goal point and the vertices of obstacles in the C-SPACE. All nodes which are unobstructed from each other are connected by straight line segments, which define the map. Obstacles in the visibility graph representations of environments are denoted as polygons were the environments are continuous or discrete spaces.

The problems encountered when employing the visibility graph are, firstly the size of the representation and the number of nodes increase with the number of obstacles, which increases the nodes and thus the complexity. The method is fast and effective in environments with low obstacle densities, but is slow and ineffective in cluttered environments.

The second problem is that solution paths take the robot very close to the obstacles as possible, while navigating from the initial node to the goal node. It is possible to show in that the shortest path on the visibility graph is optimal in terms of length. This result means that safety concerns are foregone for optimality. A solution would to extend the representation of the dimensions of obstacles outward. This keeps the actual obstacle vertices at an arm's length, so to speak from the robots radius and obviously losing the optimal length results of a conventional visibility graph representation [8].

2.2.2 Voronoi diagram method

In contrast to the visibility graph method described above, the Voronoi diagrams have a tendency to maximize the distance between the robot and obstacles (also polygons) in the map, in the sense retracting away from obstacles. Voronoi diagrams are constructed in the way that a line segment is drawn equidistant from the sides of the two nearest obstacles on each side of the same line segment. Then all these line segments are connected together at ridges/edges, which indicate the maximum distances away from the nearest obstacles collectively. A path planner can then follow the line segments turning at ridges to navigate from a start point to a goal point [8] [19].

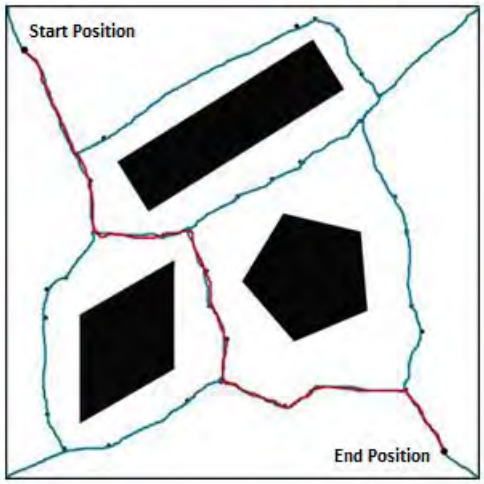

Figure 2.4: This figure shows a Voronoi diagram. The Voronoi diagram consists of lines constructed from all points that are equidistant from two or more obstacles. The goal and start positions are mapped into the Voronoi diagram by drawing a line segments from the boundary of the diagram through the goal and start points, which represents the shortest length to the ridge of the nearest line segments encompassing the nearest obstacles.

The disadvantages associated with the Voronoi diagram method are that firstly, since Voronoi diagrams are far from optimal since, in terms of total path length by very nature because of the properties of the positions of the edge and line segments in the diagrams. Secondly, since they maximise the distances to obstacles, any short range sensor is more likely to fail to sense it surroundings, if short range sensors are used for making the chosen path rather poor. There is a benefit though of the Voronoi diagram, in that a simple control system can be used navigate a Voronoi edge in a real world environment, with a laser range finder or ultrasonics [8].

2.2.3 Exact cell decomposition

In the exact decomposition method the free space, in the C- space is divided into smaller regions called cells. This is essentially seen as drawing vertical line segments at the horizontal coordinates of the beginning and end points of all the obstacles to divide the space up, then allocating numbers to all the resultant free space regions. The main idea is that the position of the robot does not matter as much as the robot's ability to move from each free cell to its neighbouring free cells. This results in a connectivity graph like the one seen in Figure 2.5(b) below [20].

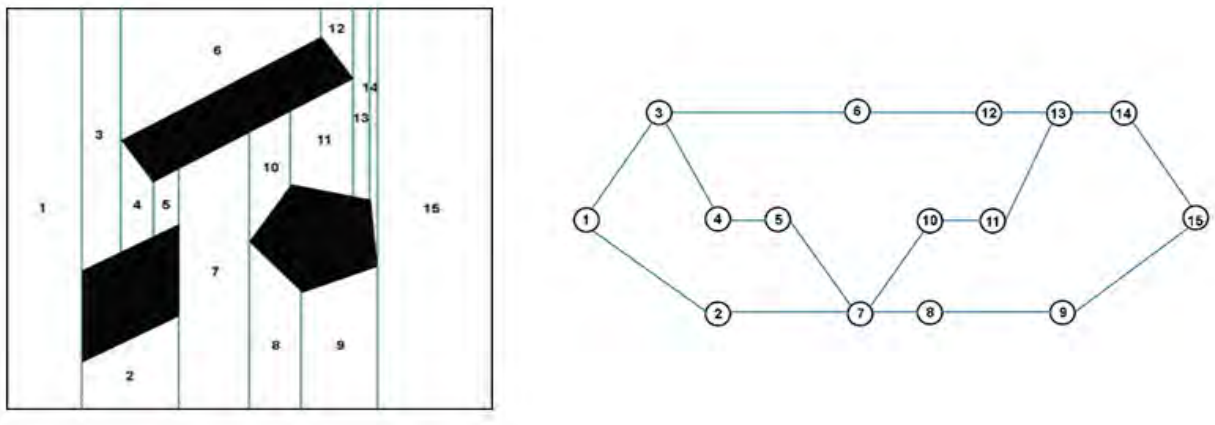

Figure 2.5: This figure shows an illustration of exact cell decomposition (a) and the resultant connectivity graph connectivity graph (b).

The main problem associated with the exact call decomposition method is that, overall planning efficiency depends on the density and complexity of obstacles in the environment. A favourable advantage is that uncluttered environments result in a lower number of cells, even if the environment is geometrical very large, which is very efficient. However due to intricacies involved in execution of the method, it is rarely used in contemporary mobile robots applications.

2.2.4 Approximate cell decomposition

The difference between approximate cell decomposition method and exact cell composition introduced above is that, a recursive method is used to continue subdividing the C-SPACE until each and every cell either lies in free space or in an obstacle region. It can also halt subdividing when it reaches a random resolution limit. The cells are subdivided into a grid structure. Each cell is divided into four smaller cells of the same shape each time it gets decomposed. After the decomposition step is complete a path can be followed through free neighbouring already decomposed cells [8] [20].

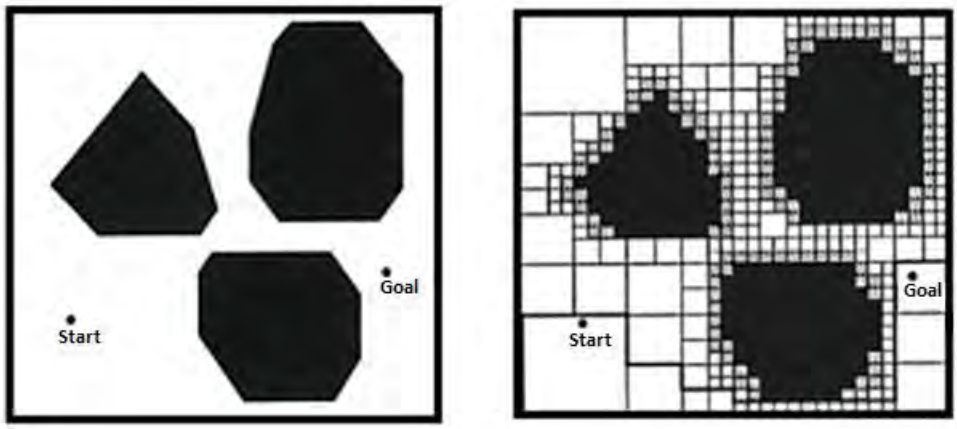

Figure 2.6: This figure shows an illustration of a C-SPACE before approximate cell decomposition (a) and C-SPACE after approximate cell decomposition (b) (Adapted from [20]).

The pronounced advantage of approximate cell decomposition is the low computational complexity [8].

2.2.5 Occupancy grid

An occupancy grid also known as a fixed cell decomposition, is similar to approximate cell decomposition, in that it is divided into a discrete grid structure as well but, differs in that the entire C-SPACE is divided into smaller cells of equal dimensions. Each cell is distinguished by whether the cell is empty (free space) or whether the cell is completely (or partial filled) by an obstacle. This method works well with robots equipped with range sensors, because each sensor when coupled with the absolute position of the robot can be used to update the filled or empty values of each cell in the C-SPACE representation of the environment [8].

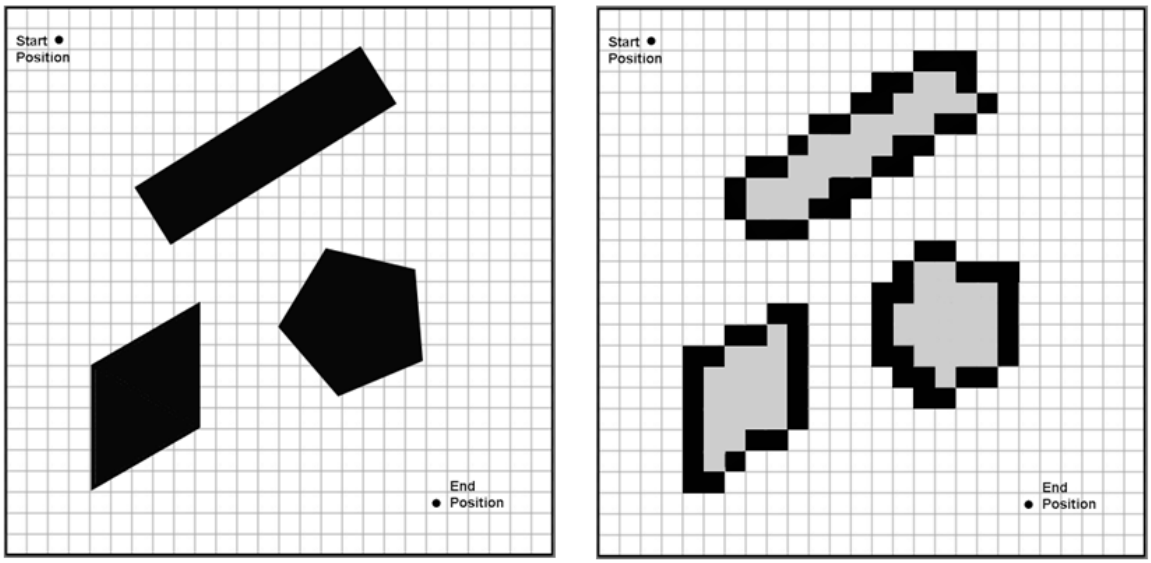

Figure 2.7: This figure shows a C-SPACE of equally divided cells (a) and the same C-SPACE after fixed cell decomposition (b).

The main disadvantages of the method are firstly, because the dimensions of the C-SPACE in the robot memory grows with the size of the environment and since the occupancy grid must have memory set aside for each every cell in the grid, the size of required memory can quickly become too large if the environment involved is large. Secondly if the geometry of the environment is not the most useful feature, then the method can prove unsuitable [8].

2.3 Graph search methods

In the previous section the author discussed the methods of constructing graphs. Path planning algorithms used in resultant connectivity graphs mentioned above-spaces are known as graph search methods. The goal of a graph search method is to find the best path i.e. the shortest path, in the connectivity graph from the start location to the goal location in no matter what map representation is chosen to generate the connectivity graph. In the following sections several graph searching algorithms will be discussed [8].

Before investigating different graph search methods, it's important to introduce some concepts, such as the expected total cost $f(n)$, path cost $g(n)$, edge traversal cost $c(n, n')$, and heuristic cost $h(n)$, which are all functions of the node n and the neighbouring node n'. Firstly the accumulated cost from the start node to any specified node n is represented as $g(n)$. Next is $c(n, n')$ which represents the cost from node n to an adjacent node n'. Lastly the expected heuristic cost from node n to goal node is termed $h(n)$.The total expected cost is then written as shown below in Equation 2.1 , where ε is a parameter that depends on the particular search algorithm the effects of which will be elaborated on in the following sections.

$$f(n) = g(n) + \varepsilon \cdot h(n) \quad (2.1)$$

2.3.1 Breadth first search

The breath first search (BFS) method is a graph-search algorithm which begins at a start node (position), and explores all its neighbouring nodes. The neighbouring nodes explore all their neighbouring nodes, and so on, until the goal node is marked as visited. In other words, it systematically searches the entire connectivity graph until it finds the goal node. In the BFS method each individual edge is assumed to have the same cost (like in an occupancy grid), hence an optimal plan solution can found in simpler implementation than other graph search algorithms as well as to obtain faster speeds [8].

To elaborate further, starting from the start node or parent node, lateral neighbouring nodes are first expanded based on the proximity to the start node, this means choosing adjacent nodes which can be accessed by the shortest number of edge transitions. Then afterwards row by row until the goal node is reached where it ends. The computation of a solution is faster because tracing back the solution, nodes are already sorted by increasing vicinity to the start node, and hence the search always returns the path, with the least amount of edges from the start node to the goal node [8] [21].

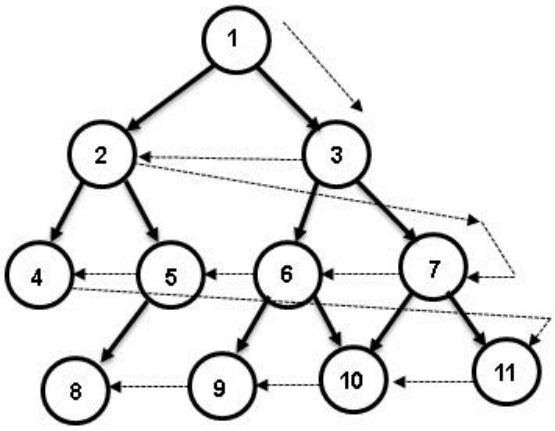

Figure 2.8: This schematic illustrates the Breadth first search technique, how the BFS method expands from the parent node to the goal node, layer by layer, where 1 is the start node) and 8 is the goal node.

It is an important aspect to Figure out whether the algorithm will output a cost-optimal solution. If the costs of all individual edge transitions are the same, the BFS will always return a minimum cost part. However if the node transition costs are non–uniform, there is no assurance of the return of a cost-optimal path. This means the path with the lowest number of edges does not guarantee cost effectiveness. A path with more edge transitions could in fact produce a lesser total cost [8].

The entire connectivity graph can be stored in memory, but the complexity of the search algorithm is determined by the number of nodes actually expanded upon on the way to the goal rather than the size of the environment or the density of obstacles within said environment [8].

2.3.2 Depth first search

The depth first search (DFS) in distinction to the BFS algorithm discussed previously expands each node until the lowest (deepest) layer of the connectivity graph. The algorithm then goes to the next available highest adjacent node expands until the lowest level. When each node is fully expanded upon, that branch or section is removed from the search and the highest unvisited node is selected and the process is repeated until it terminates at the goal [8].

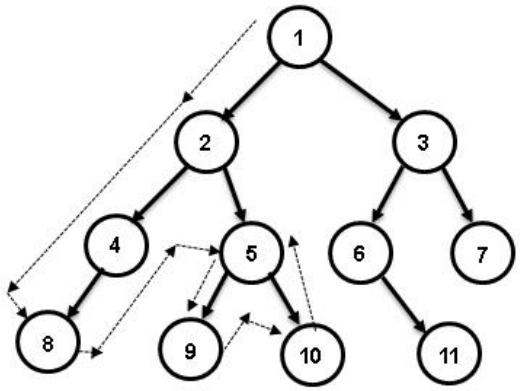

Figure 2.9: This schematic shows an illustration of the depth first search method

A disadvantage of this method is that it may return to previously visited nodes or enter truncated paths. This can be avoided by an efficient implementation. An advantage is that the algorithm, in terms of space complexity, is that it only needs to store one path from the start node to goal node. Once a node and all its children have been visited without reaching the goal node in the process it can be removed from memory. Just like the BFS algorithm the non-uniformity of individual costs affects whether the returned path will be cost optimal [8].

2.3.3 The Dijkstra's algorithm

The Dijkstra's algorithm named after E.W.Dijkstra its inventor is comparable to the BFS introduced previously in section 2.3.1. However, the difference in edge costs does not affect whether the path returned will be cost optimal, as long as the edge costs themselves are positive values. In order to obtain this consistent optimality, some complexity has been added in the form of a concept called a heap or a tree based search structure [58].

The big difference in the algorithms is, initially Dijkstra's algorithm expands nodes from the start node similarly to the BFS method, but the neighbours of that particular visited node are place in a heap and sorted according to their expected total cost $f(n)$ values introduced previously above. This is effectively the $g(n)$ value as there is no heuristics $h(n)$ used in the method. The nodes are sorted from lowest total cost to highest total cost. This means the node with the lost cost is sent to the top of the heap and accessed and expanded and so on, until the goal node is visited or no nodes remain in the heap. The optimal solution path can then be found by backtracking from the goal node to the start node [8].

The benefit of the algorithm is that not only the optimal path is calculated, but also all the other cost effective paths from any start node to goal node in the environment map. This enables a robot to reach any goal in the environment without having to start the process again.

2.3.4 The A* algorithm

The A* algorithm pronounced "a star" is comparable to Dijkstra's algorithm, except for the heuristic function $h(n)$ included in calculating the total expected cost as shown by Equation 2.2 in below, similar to Equation 2.1 . $g(n)$ is the cost of the path from the initial state to node n and $h(n)$ is the cost or a path from node n to a goal. Hence, $f(n)$ approximates the lowest total cost of any solution path going through node n. The particular heuristics function $h(n)$ for this algorithm requires some supplementary information about the graph. The edge costs need to be undervalued in order to guarantee solution optimality, which is what the heuristic function achieves.

$$f(n) = g(n) + h(n) \quad (2.4)$$

The A* algorithm begins at the start node and puts all its neighbours in a heap, which is then sorted from lowest $f(n)$ value to largest, the one difference being that the heuristic function values are included in the total expected cost. The lowest cost is then accessed and expanded and so on, until the goal node is visited. If there are nodes of equal $f(n)$ value; out of them, the nodes with the lower h values are favoured [103]. The cost-optimal solution can then be obtained once again by backtracking from goal node to start node [8]. The implementation of this algorithm usually provides a better performance than that of the Dijkstra's algorithm. A* algorithm only finds an optimal path to a goal if the heuristic function $h(n)$ is admissible; meaning it never overestimates actual cost of a node [103].

In most robotics applications A* is used with an occupancy grid (introduced previously in section 2.2.5). The heuristic elected to be used with this research endeavour is the

distance from any cell to the goal cell. This actively reduces the number of node visitations required to arrive at a solution, unlike Dijkstra's algorithm.

2.3.5 The Q-learning algorithm

Q-learning is an off policy model free algorithm introduced by P. Watkins 1989 which does not require explicit knowledge of the environment [10]. It allows an agent to form and improve behaviors through trial and error. The objective of a Q-learning agent in this Markov decision process is to learn a policy π by sensing the current state or node at each discrete time-step t and selecting an appropriate action .The environment then changes from its current state s_t to the next state (node) s according to the state transition probability function given by f. Then a scalar reward of r_t is given right after the transition [22] [23]. The Q-learning algorithm will be explained in much finer detail in the conceptual framework in Chapter 3.

2.3.6 Important considerations

Out of all these algorithms discussed earlier the author observed that some of the concepts introduced, are useful in the development of a Q-learning based path planner essential for this study. Firstly the occupancy grid decomposition method is used as a standard C-SPACE representation for mobile robot applications [23]. Secondly the non-uniformity of edge cost could be realised in environments where obstacle position effectively changes the total expected cost of a node e.g. the obstacle is directly in line with the current location(node) and the goal location, if the heuristics are distance based heuristics like suggested in the A* implementation, hence undervaluing particular edge costs. Lastly the concept of backtracking is also used in Temporal Difference (TD) implementations which are the family of algorithms from which the Q-learning algorithm was developed [24]. Other algorithm such as the D* algorithm and the Potential Field planning algorithms will not be discussed as their literature is considered outside of the scope of this study since, the study is to do with static environments and using a

occupancy grid respectively. The Randomised graph search algorithm [8] is an algorithm to consider when dealing this the greediness of action selection policy. It is introduced in the next chapter and is not huge factor in the next section and so will not be elaborated upon any further, besides pointing out that the method's name describes the process abundantly enough.

3 Q-LEARNING

In order to better understand the Q-learning algorithm it essential to explore the literature from which Q-learning originated i.e. the family of algorithms called Reinforcement Learning (RL) algorithms which is an area of interest in the field of Machine learning. It is important to note that Q-learning is used to solve a path planning problem in this study but has various other applications. Firstly this conceptual framework will briefly introduce Machine learning and its different groups of algorithms. Secondly, the framework will give justification why out of the types of Machine learning algorithms, the author chose the Reinforcement learning algorithms. Thirdly, this will be followed by a brief overview of reinforcement learning. Next, a justification will be made why the Q-learning algorithm was chosen out of the types of Reinforcement learning algorithms. Lastly the relevant literature concerning the Q-learning algorithm will be presented.

3.1 Machine Learning

Machine learning is a core research area of Artificial Intelligence (AI) that pertains to the development of computer algorithms and techniques that are able to learn or improve automatically through experience [25] [26].

The Machine learning algorithms generally consist of the following steps [27]:

1. Selecting a candidate model for the learning problem.
2. Approximating the parameters of the above model using a learning algorithm and obtainable data [26] .

Often, the above mentioned steps are iterated in the learning algorithm. The model selected is normally chosen intuitively and is based on observation.

Machine learning algorithms can generally be separated into three groups of learning algorithms, namely supervised learning, unsupervised learning and reinforcement learning algorithms. Supervised learning is learning from examples provided by a supervisor or teacher.

Secondly, in the unsupervised learning group, there is no teacher, only input data. The focus is to study how systems can be used to represent particular input patterns in a way that reflects the statistical structure of all the input patterns. Finally in reinforcement learning, the goal is to uncover a deterministic scheme or decision making policy by which an agent (the control system of the robot) can select actions that maximize the long term cumulative rewards given through interaction with an environment for the purpose of accomplishing a task. The next section narrows the focus, through the introducing of a conceptual framework.

3.2 The reasons for using reinforcement learning algorithms

Due to the unknown and unstructured nature of most environments, supervised learning is not suitable for learning by interaction, because in interactive problems it is impracticable to obtain examples of desired behaviours that are correct and represent all the situations in which an agent has to act. On the other hand reinforcement learning agents learn by receiving a reward or reinforcement from the environment, without any form of supervision other than its own decision making policy.

In terms of unsupervised learning the agent has to first collect data from the input and find patterns in the input data to make decisions from. This is difficult to implement because of the Computer processing unit and memory constraints of most available robots computers as well as that collecting data is time consuming. Reinforcement learning differs in that, as soon as an optimal policy if obtained through training or simulations, the policy can be implemented straight away in a real environment [28] [29]. Bearing this in mind it is important to note that, RL simulations also require knowledge of the environment and can consume a lot of computational resources.

3.3 The reinforcement learning model

The first concept to introduce is that of reinforcement or reward. A reward is given to an agent in return for each action taken. Figure 3.1 below demonstrates this concept. Firstly, the agent receives a representation of the environment known as a state. In Figure 3.1 s_t is an element of the set of all possible states S i.e. ($s_t \in S$). For example a state in an environment such as a physical location can be represented by coordinates; the agent selects an action α_t out of all possible actions ($\alpha_t \in A(s_t)$), based on a particular policy.

Following the action, on the next time-step the agent receives a numerical reward $r_{t+1} \in R$, as the consequence of the chosen action. The r_{t+1} is a delayed reward which is used primarily for the general model of reinforcement learning and will be discussed in a subsequent section in more detail. The rewards can be physically represented by signals passing from the environment to the agent, or in this context a robot. The agent then may make a transition to a new state s_{t+1} .This process can be observed to some the agent's. The process iterates until goal of the agent is achieved

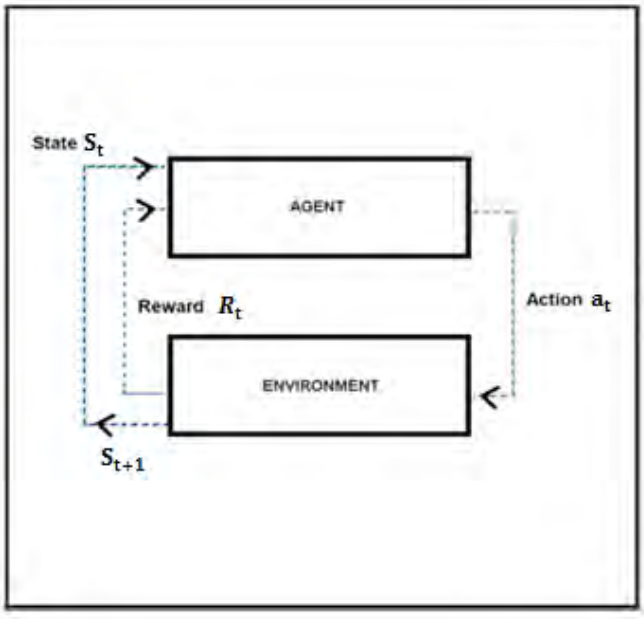

Figure 3.1 This figure shows a basic model of the Reinforcement Learning system [30]

The reward is represented as a numerical value; $r_t \in R$ .The objective is thus to maximize the sum of rewards over all the time steps of an episode. An episode is defined a single complete execution of all the steps of an algorithm. This cumulative total is typically known as the total reward or return R_t shown below in Equation 3.2 [24]. The Equation of the return can be seen below in Equation 2 where r_τ is the reward coinciding with $\mathrm{t} = \tau$, which is the final time-step.

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots r_\tau \quad (3.2)$$

Another important concept introduced at this point is that of discounting where the effect of the value of future rewards on the expected return sum can be reduced or increased depending the value of a parameter γ, which is chosen between $0 \leq \gamma \leq 1$. γ is called the discount rate or factor. The closer γ is to 1 the more strongly future rewards are taken into account. The discount rate determines the present value of future rewards: a reward received k time steps in the future is worth only γ^{k-1} times what it would be worth if it were received at the current time-step [24]. Equation 3.3 shows the discounting is called the expected return sum E [24].

$$E = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k \ r_{t+k+1} \quad (3.3)$$

3.3.1 Rewards

The agent's goal in reinforcement learning problems is typically to maximize the total amount of reward it receives, which means not just maximizing the immediate reward, but expected return in the long term. The learning system will not converge to a good policy that achieves the task, if the immediate rewards are not allocated greedily enough. The action selection strategy must be partially deterministic in order for the resultant policy obtained to converge to be a good policy. This is called the temporal credit assignment problem.

The degree of delay in the reward signal creates three different classes of reinforcement learning problems such as:

1. Immediate reward
2. Delayed reward
3. Pure-delayed reward

In immediate reward problems the reward is returned immediately in the current time step and completely describes the value of taking an action in a particular state. In this class of problems only the current state is important, and there is no need to allocate a reward for a sequence of future actions. In fact, these problems can be solved by setting the discount factor to zero. Immediate reward problems are uncommon since actions typically take time to affect the environment [31].

Delayed reward problems span a wider range of tasks. In this class of problems there may be a reward available at every time step of an action in a particular state. Robotics problems often require delayed rewards. In these problems it is necessary to carry rewards back through time; therefore discounting factor must be greater than zero [31].

In pure-delayed reward problems the reward at each step of an episode is the same, except possibly the last step, when a terminal reward is given which indicates success or failure. They are generally set this way in order to facilitate the achieving of sub goals; however, importance is placed on achieving the main goal rather than just sub goals. In these types of problems the discounting factor is generally required to be close to one so that the reward can be carried back over many steps [31].

3.3.2 Markov decision making processes

In order to allocate the rewards adequately it is necessary to see how the environment is perceived to the agent. The concept of a state which was mentioned above is usually denoted as the agent's perception of the environment. A state can be seen as vector combination (tuple) of different sensor measurements. A state signal or vector that contains all relevant information about the environment is said to be Markovian. A reinforcement learning problem whose state vector is deemed to be Markovian can be handled by Markov decision processes (MDP), which according to M. Puterman [32] "are models for sequential decision making when outcomes are uncertain".

Z. Zhongli *et al.* [22] present a MDP model which can be characterised by the following four-tuple vector (S, A, R, P). In the model S is environment state set, A is system action set, $R: S \times A \rightarrow R$ is reward function and P: $S \times A \rightarrow P$ is a state transition probability [22, 32, 33], $R_{ss'}^{a}$ is a function defined as "the instantaneous reward value obtained by system when the environment state s changes to state s' through action a" and $P_{ss'}^{a}$ is a function defined as the "probability obtained by system when the environment states changes to state s' through action a".

As mentioned above the state vector which contains only the necessary information is said to be Markovian. In terms of the discussed MDP model, this means that present reward r and subsequent states s', are dependent on current state s and action a only and not on any previous sensor measurements .This property is known as the Markov assumption and is presented below where the state transition probability function $P_{ss'}^{a}$ maps state s and action a to state s$'$. The reward function $R_{ss'}^{a}$ maps state s and action a to reward r.

***Markov assumption*:** $s' = P_{ss'}^{a}(s, a)$ and $r = R_{ss'}^{a}(s, a)$

Sometimes the $P_{ss'}^{a}$ and $R_{ss'}^{a}$ functions may not be known to the agent, so the system initially opts for a policy corresponding with an instantaneous reward resulting from trial-and-error [22]. This involves having to analyze the uncertainty of the MDP model and considering long-term goals in order to select appropriate strategies. The typical variances to the MDP model depend on the perception of the states i.e. totally observable (accessible) or partially observable (inaccessible) [34].

3.3.3 Value functions

If we proceed with the assumption that the environment we will be dealing with are Markovian enough to be dealt with by a MDP, a natural progression is to estimate how good it is for an agent to be in a particular state. For this express purpose, the concept of the *value function* will now be introduced [34]. A value function is a function of state or state-action pairs and is in fact a form of an expected return as introduced in section 3.3

The value function shown in Equation 3.4 below is effectively an expected return corresponding with the agent following a particular policy π. $V^{\pi}(s)$, the value function in Equation 3.4 below is defined as the expected return E_{π} when starting in s and following policy π from then on [34]. t is any time-step. Similarly $Q^{\pi}(s,a)$ in Equation 3.5 is a value function defined as the expected return E_{π} starting from s , taking the action a , and thereafter following a policy π .Note that the value function can be function of both state and action, also known as state-action pairs. [32, 22, 35].

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\} \quad (3.4)$$

$$Q^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s, a_t = a\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\} \quad (3.5)$$

In order to improve the usefulness or the utility of value function, numerous policies have been developed. For example, the greedy action selection policy always chooses the action with the maximum estimated reward. At times the perceived action with the maximum reward may be a suboptimal choice due to sampling errors or the agent being forced to explore only perceived maximum reward states and not all states, causing the agent to sometimes miss optimal states. This is known as the struggle between exploration and exploitation.

This can be remedied by choosing a policy that allows the agent to explore more of the environment in other to possibly find more optimal actions which might yield a larger return over time. For example the ε-greedy action selection policy is slightly different from the greedy algorithm mentioned above as it chooses a random action according to a small exploration probability ϵ_p and the rest of the time it choose an greedy action with a probability of *1-*ϵ_p exploitation [23].

3.4 Types of reinforcement learning

3.4.1 Model based algorithms

In Model based algorithms the agent attempts to learn the model of its environment, which allows it to anticipate the consequence of actions before they are even executed, meaning the agent can generate what is called “virtual experience” and so a search through its model of an environment to find and optimal solution. The model can then be searched in any direction which makes it “motivationally flexible”, meaning the agent can search the model in a particular direction depending on unique goals. In terms of the MDP introduced above it can mean learning the $\mathrm{P}_{\mathrm{ss}'}^{\mathrm{a}}$, and $\mathrm{R}_{\mathrm{ss}'}^{\mathrm{a}}$, functions mentioned in section 3.3.2. The advantage of model based algorithms is that, learning is simple as there is no “temporal complexity”. This means every piece of knowledge that can be learnt at just the right time as there is instant feedback from the environment, which makes it statistically efficient. The clear disadvantage is that the knowledge learnt is difficult to implement because it means

searching through vast structures in the memory allocated to the model, making it "computationally intensive" [36].

3.4.2 Model free algorithms

Model-free algorithms also known as temporal difference algorithms mainly feature value functions shown as below in Equation 3.6. This means the algorithms can successfully minimize inconsistency between successive predictions of the value of the position of states in an environment. Equation 6 can be manipulated to obtain what is known as the temporal difference error signal or training signal as shown in Equation 3.7. This error can be used for training, which means interacting with the environment through actions and building a value function, hence removing the need for a model.

The advantage of model free algorithms is that they are computational appealing as they are easy to use and direct access to the value of making certain decisions, is inherent in the value functions like the one shown in Equation 3.7 below. This is the same value function introduced above in section 3.3.3. The agent in these model free algorithms are generally "motivationally insensitive" as the agent will not be aware of the actual rewards it can get, but rather is only aware that it will receive a numerical value for each state transition. The clear disadvantage is that there are prediction errors associated with future predictions, because of minimizing inconsistency. The $R^{a}_{ss'}$ function is not based on good quality information because rewards are unchanging numerical values which do not necessarily match a possibly changing world or dynamic environment. Hence model free algorithms tend to be "statistically inefficient" [37].

= =

3.5 Justification for choosing a model free algorithm

The setting that this thesis is based on is that of an industrial environment. An industrial environment can be dynamic or at least partially dynamic. Which means that any model or the environment built up for the agents use could change from moment to moment causing unanticipated errors in the model. This coupled with the computational intensiveness of have to search through entire model to compare with actual environment. Model free algorithms tend to be more suitable, because they are set up to minimize inconsistency in perception of states, through actual experience. Despite being statistically inefficient, they more than make up for it by being computationally easier on the agent. This is favourable because even if the environment changes the algorithm can learn an optimal policy for a new environment quicker than it takes to build a new model for the environment.

3.6 Justification for choosing the Q-learning algorithm

The Q-learning algorithm was chosen to utilize in this research project mainly because of its accessibility. As indicated earlier, Q-learning has been widely researched and the theory has been refined considerably. It has also being found to typically be easier to implement than most of the other model-free or temporal difference type algorithms. Q-learning is also an appealing method of learning because of the ease of the computational demands per time step [38], and also because of this proof of convergence to a global optimum, avoiding all local optima as long as the environment concerned is a Markov Decision Process, which was introduced in an earlier section.

Q-learning is an off-policy algorithm, which means training data for the learning processed can be accumulated faster. An off-policy algorithm learns the value of the optimal policy independently of the agent's actions during the training phase [102]. Simulations frequently deliver results faster than online implementations and can be modified easily. It can also be used in both discrete and continuous environments. Theory supports the ability to converge to an optimal policy provided enough exploration of the environment [39] [40].

3.7 Q-learning algorithm

To illustrate Q-learning consider a finite MDP model $< S, A, f, R >$, similar to the one presented in section 3.3.2 above. In this model, S is a finite discrete set of states, A is a finite discrete set of actions, $f : S \times A \rightarrow \Pi(S)$ is the state transition probability function and $R : S \times A \rightarrow \boldsymbol{R}$ is a reward function. It is helpful to remember at this point that although f and R exist, they are not necessarily known by the agent. Even within the MDP model construct there are still different categories of learning scenarios namely "deterministic worlds" and "non-deterministic worlds".

3.7.1 Deterministic worlds

In the deterministic worlds the reward function R can be known to the agent. The corresponding value function $V^{\pi}(s_t)$, containing the rewards at each time step can be obtained by following a policy π chosen arbitrarily from any initial state s_t . The value function $V^{\pi}(s_t)$ is shown below in Equation 3.8, r_t is the reward at current time-step t and γ is the discounting factor.

$$V^{\pi}(s_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i} \quad (3.8)$$

In this MDP an optimal policy $\pi *$ is necessary to obtain maximum value function return. Equation 3.9 below shows the policy $\pi *$, where $V^*(s)$ is the value function. This Equation can be further broken to reveal Equation 3.10, which is derived from performing what is called a "look ahead search", essential looking ahead to choose the best action from any state s. This is only possible if the agent knows the reward function R and the state transition probability function f.

$$\pi^*(s) \equiv arg \max_a V^*(s) \quad (3.9)$$

$$\pi^*(s) \equiv arg \max_a [r(s,a) + V^*(f(s,a))] \quad (3.10)$$

To solve the problem of not having f, a new function $Q(s,a)$, which is similar to $V^{\pi}(s_t)$ in Equation 3.11 can be defined. The agent just has to learn $Q(s,a)$ after which, it can choose optimal action without knowing what function f is. So if $Q(s,a)$ is learnt, $\pi^*(s)$ gets modified to Equation 3.12.

$$Q(s,a) \equiv r(s,a) + \gamma V^*(f(s,a)) \quad (3.11)$$

$$\pi^*(s) \equiv arg \max_a [Q(s,a)] \quad (3.12)$$

It is essential at this point to note that V^* is closely related to Q as shown below in Equation 3.13, which means Q can now be written recursively as shown in Equations 3.14 and 3.15 below. The current approximation of Q, is called $\widehat{Q}$, and is the training rule used to train the agent. $\widehat{Q}$ is displayed in Equation 3.16 ,where s' is the state at the next time step and a' is now known as an action out of all possible action in a set of actions belonging to the next state s'. The $\widehat{Q}$ values can then be stored in a table, from which the optimal policy can be retrieved when needed by the agent.

$$V^*(s) \equiv arg \ max_{a'} Q(s,a') \quad (3.13)$$

$$Q(s_t, a_t) \equiv r(s_t, a_t) + \gamma V\big(f(s_t, a_t)\big) \quad (3.14)$$

$$Q(s_t, a_t) \equiv r(s_t, a_t) + \gamma \ max_{a'} Q(s_{t+1}, a') \quad (3.15)$$

$$\widehat{Q}(s,a) \equiv r(s,a) + \gamma max \ Q(s',a') \quad (3.16)$$

3.7.2 Nondeterministic worlds

In the case of nondeterministic worlds the reward function R and transition function f are nondeterministic. This means there is some uncertainty in relation to the obtained value of the next reward or the next state cannot be predicted with a probability of 1 or 0. In this case the solution is the redefined variables V and Q in terms of expected values as shown below in Equations 3.17 and 3.18 below [34].

$$V^{\pi}(s) = E[\, r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \,] = E[\sum_{k=0}^{\infty} \gamma^k \; r_{t+k+1} \,] \quad (3.17)$$

$$Q(s,a) \equiv E[r(s,a) + \gamma V^*(f(s,a)] \; (3.18)$$

The decaying factor or the learning rate α is now introduced. This alters the deterministic world training rule to the form shown in Equation 3.19 below. $\alpha = 1/(\mathbf{1} + n(s,a))$, where $\alpha \in [0\colon 1]$ and n is total number of times (s,a) has been visited. Please note that the reward r can also be stochastically determined. $\widehat{Q}$ converges to Q in a nondeterministic world similarly to the deterministic world case when (s,a) are visited infinitely often.

$$\hat{Q}_n(s,a) = (1-\alpha_n)Q(s,a) + \; \alpha_n \, [r + \gamma max \, \hat{Q}_{n-1}(s',a')] \; (3.19)$$

3.7.3 Pseudo code for Q- learning algorithm

The following is pseudo code for the Q-learning algorithm [41], the only thing that changes in the case of the deterministic and nondeterministic worlds are the update Equations for the Q values.

1. Initialize Q(s, a) arbitrarily
2. Repeat for each episode
   - 2.1 Initialize s arbitrarily
   - 2.2 Repeat for each step of the episode
     - 2.2.1 Choose a from s using policy derived from Q
     - 2.2.2 Take action a and observe r and s
     - 2.2.3 $Q(s,a) = Q(s,a) + \alpha[r + \gamma \times \max \times Q(s',a') - Q(s,a)]$
     - 2.2.4 $s = s'$ until s reaches goal state

     End

4 RELATED STUDIES

As mentioned in section 1.3 above, the purpose of this study is to evaluate the effectiveness of the Q-learning algorithm in producing energy efficient paths, within occupancy grid decomposition. To this end, several related works were perused with the intention to obtain some insights which could be used to construct the experiments needed to achieve the intended goal.

C. Watkins *et al.* [10] assert that Q-learning is a simple way for agents to learn how to act optimally in controlled Markovian domains. They have also asserted that Q-learning converges to the optimal action policy as long as all the actions are repeatedly sampled in all states and the action are represented discretely. This view is derived from Watkins (1989) PhD thesis titled, "*Learning from Delayed Rewards."* C. Watkins *et al.* (1992) provide a thorough proof to support their claim [10].

According to L. Kaebling *et al.* [23] Q-learning uses a method which is essentially the same as the *temporal difference method* or TD (0) developed by Sutton in 1988. While discussing the same subject of the convergence of the Q-learning algorithm, they suggested that when Q-values are nearly converged to their optimal values, they are apt for actions to be executed in a greed manner in every instance thereafter. This is so as to obtain the highest Q-values. L. Kaebling *et al.* also state that Q-learning is "exploration insensitive", meaning that the Q-values will converge to optimal values independent of the action policy used during the exploration phase [23].

L. Zamstein *et al.* [63] used the Q-learning algorithm to perform the initial learning for a mobile robot butler named "Koolio", as detailed in the paper titled *Koolio: Path planning using reinforcement learning on a real robot platform***.** This algorithm was chosen because of the inherent temporal difference structure of the algorithm, which allows the learning agent to handle changes in the environment and bootstrapping. L. Zamstein *et al.* defined bootstrapping as "The method used to estimate the state and action values based on

estimates of later stages" [63]. Bootstrapping is also known as "backups" from other works such as Bersekas and Singh works in 1989 and 1993 respectively [23] [63].

L. Zamstein *et al.* also assert that the episodic nature of reinforcement learning leads to a large number of repetitions, which if applied directly on a mobile robot it could cause wear and tear. The solution that they have also suggested is to undertake the initial learning stage in simulation. They further claim that a reinforcement learning process only functions properly, if the said process is assumed to be Markovian in nature, which is in line with the view of C. Watkins *et al.* [63] as well.

It is apposite to state that the paper has failed to mention how to solve the problem of imperfect or incomplete perception of the states of the environment. L. Kaebling *et al.* provide possible solutions to deal with these noisy and incomplete perceptions [23]. Accordingly, L. Zamstein *et al.* have suggested the use of an e-greedy algorithm for the training stage of learning. However, they do not mention of how to deal with the complexity caused by dense or complex obstacle structures in an environment. They also do not seem to indicate which graph construction method was used in the simulations or how bootstrapping was achieved. Therefore, it will be difficult to replicate the experiment.

Contrastingly in the paper titled *Q Learning for Mobile Robot Navigation in Indoor Environment, D.* Tamilselvi *et al.* propose that the learning method should be treated as two subjects namely, the learning mobile robot and the learning environment [48]. To them, "At successive time steps, the agent makes an observation of the process state, selects an action and applies it back to the process; thus modifying the state. The goal of the agent is to find out adequate actions for controlling this process," [48].

What is of interest in the paper is that, D. Tamilselvi *et al.* make use of a grid environment for their simulation and store Q-values in a Q-lookup-table [48]. The paper is also relevant to my study because the same form of graph construction method has been adopted by the author for this research project. The Q-lookup-table is essential because it has been proved to guarantee convergence for the Q-learning algorithm [10]. A major factor of concern in the

paper though, has been the use of the learning rate parameter. D. Tamilselvi *et al.* seem to have obtained results which indicate that by increasing the learning rate the convergence time decreases. This is in conflict with what was stated by C. Watkins *et al.* and L. Kaebling *et al.* [10][23].

Q. Zhang *et al.* [68] also used a grid environment for their simulations, in their paper titled *Reinforcement Learning in Robot Path Optimization.* The simulations used a delayed reward system, which only allocates a huge reward at the end of an episode run. This approach is of interest to this study whose goal is to reduce the total energy costs for the entire path rather than just individual discrete costs. The Q-learning algorithm was tested in both aggregated and scattered environments, within deterministic and non-deterministic MDP constructs. The results are illustrated clearly enough for one to see how effective the Q-learning algorithm is at handling these types of environment representations [68].

Y. Li *et al.* [46] proposes a novel method to increase learning convergence speed. The method is called the Priority Q search algorithm and uses the sum of weighted vectors pointing away from obstacles to allocate the rewards and punishment. The reinforcement values can then be predicted from the vectors that are more likely to achieve the goal and given a rank. The agent is then more likely to pick the higher ranked actions to execute during navigation. This method provides an alternative to the other implementations of Q-learning, while still using the grid environment representation. What is evident from the results is that an optimal path is produced, which goes through the region with the least obstacles density. This finding result is important because this setup is more likely to conserve energy due to the reduced obstacle avoidance needed [46].

5 MINIMISING THE ENERGY CONSUMPTION OF MOBILE ROBOTS

There has been an increasing concern for energy usage over the last few decades, due to the many developments of mobile robots. Since most contemporary mobile robots are powered with batteries, the running life of the robot is finite. This has stemmed an interest in mobile robot energy related practices. Section 4.3 follows up with an approach for the optimising of graph search methods, as a way to minimise energy consumption.

5.1 Defining Energy efficiency of Mobile robots

The performance of a machine can be evaluated by calculating the efficiency of the machine. Efficiency is defined as the measure of how much of the input to the machine is used to produce the output of the machine, which can be seen in Equation 4.1 below.

$$Efficiency = \frac{Output}{Input} \quad (4.1)$$

The energy stored in a battery depletes according to the rate of consumption by the mobile robots components and actuators. Since it can be costly to replace or recharge batteries and with the many current applications of mobile robots it is important to investigate appropriate energy utilization practices. These applications include the navigation of dangerous situations; ocean exploration; medical and surgical applications as well as space and sea exploration [42]. With the above mentioned in mind the efficiency can now be defined as seen in Equation 4.2 below. The Output Task for example, can be the distance travelled by the robot or the operation of the robot. The developers or designers are the ones who set the output tasks to be carried out. Energy Consumption relates to how the battery power is used to power different components of the robot. So, to increase the efficiency of a mobile robot, it is necessary to complete the output task but at the same time minimise the energy consumption to complete said task [42].

$$Efficiency = \frac{Output\ Task}{Energy\ Consumption} \quad (4.2)$$

DC motors are commonly used with mobile robots. In DC motors, the power loss comprises of armature loss, internal mechanical loss, and eddy-current loss. As the speed increase the power loss increases. The behaviour of the eddy-current loss in believed to be one the factors that lends to this trend. According to B. Kuo and J. Tal the eddy-current loss increases by the square of the speed [85]. Y. Mei *et al.* defined power efficiency as follows.

> "The motion power efficiency can be defined as the inverse of the energy per unit distance i.e. $\frac{v}{p(v)}$, where v is velocity and $p(v)$is power usage at velocity v. The efficiency will increase first as the speed increases, and then decreases due to the large power loss at a higher speed" [84].

5.2 Energy consumption in Mobile robots

There are several factors that contribute the energy consumption of mobile robots. These include amongst others, the motors, the sensors and the microcontroller. However this study is focused on energy consumed through the motion of navigating paths in environments. So the next section will be dealing with the energy consumption of the mobile robot components which use energy as a direct result of interaction with environment, for e.g. the motors.

5.2.1 Energy consumption of motors

Small mobile robots normally use DC motors as they are relatively cheap. The DC motors can be used as actuators when acting on the environment. Most mobile robots interact with their environment via wheels. The DC motors control the rotation of the wheels; hence, motion is directly dependant on the DC motors [42]. Generally DC motors supply current or voltage depending on the accompanying robot body circuitry. This indicates that battery consumption is connected to the physical amount of electrical signal required. The DC motor being a physical system also loses energy due to factors like friction and load Inertia. Another component of energy loss is due to the power consumption of the robots electronic circuit driving the motor. The power consumption of the DC motors can be defined as the sum of the mechanical output power and the transforming loss [84]. Y. Mei *et al.* propose a model for a motor as follows:

> "Let m be the robot's mass and the ground friction constant be μ. When the robot travels with a speed of v and an acceleration of a, it needs a traction force of $m(a + g\mu)$. Therefore, the output mechanical power is $m(a + g\mu)$v, where g is the gravity constant. The motion power can be modelled as a function of the speed, the acceleration, and the mass: $p_m(m, v, a) = p_l + m(a + g\mu)$v; (1) where p_m is the motion power, and p_l is the transforming loss"[84].

The speed of the motors also contributes to the power the used. Y. Mei *et al.* adequately defined the conditions under which the speed contributes to power consumed.

> "If the maximum speed of a robot is denoted as V_m , then the speed at which the robot consumes the least energy can be denoted as V_o . The most energy-efficient speed, V_o is defined as the speed at which the robot consumes the least energy to travel a unit distance, or $V_o = argmin\{\frac{P(v)}{v}\}, 0 < V_o < V_m$"[84].

5.2.2 Motion planning

This section brings attention to the effect of motion planning on overall energy consumption. In the attempt to minimize energy consumption prompts the design of "optimal" paths along which the robot can navigate. Optimality in this case here relates with any aspect of the motion of the robot that affects energy consumption [42]. The criteria for optimality in terms of path planning, which is the area of interest in this study normally refers to the amount of time spent doing a task or the distance covered during the task. It would seem intuitive for a path planner in this case to minimize the distance and/or time doing the task. However, minimizing the distance may lead to the path chosen having numerous sharp turns. Each turn would make the robot decelerate and accelerate as required by the path.

In order to generate required power for each acceleration and deceleration the battery will have to provide more voltage or current to the motor hence using more energy. So, despite the solution being optimal in distance it will not be optimal in terms of energy. If time is minimized, it will require a faster velocity which also invariable uses more energy from the motors [42]. The solution which is used in this study uses a path planning algorithm to exploit the natural existing conflicting objectives of optimising for distance or energy.

5.3 Multicriteria Optimisation for Mobile Robots

Hart *et al.* in 1968 and E. Dijkstra in 1959, both postulated that a graph search method can be made optimal by using a particular cost function, provided a graph based model was constructed for the environment in question. The cost function normally depends on a unique cost variable such as, the distance travelled or the time spent [58, 76]. Other more complex path planners would combine the above mentioned costs with other factors such as energy consumption, safety and clearance distance from obstacles. Since these individual costs were of different inherent characteristics they could not be intuitively and easily combined into a cost function, in practice. In general a cost function can be constructed with a weighted linear combination of each individual cost. It is important to note that it is not essential to optimize the value of the cost function, but to perform within the constraints placed on the cost variables.

The boundary or constraints are represented by a numerical value, which is called an aspiration level [43]. Examples of the constraints are as follows, where E can represent, for example the acceptable energy usage of a robot for a particular task and T can represent the time limit to execute the task.

Energy consumption
Arrival time $<$

Objectives in the Multicriteria framework are defined in terms of goals design as inequalities. These can be seen in the form below in Equation 4.3.

(4.3)

An example of this is as follows:

A solution path p, defined as a path that connects a pair of start and target nodes in the graph, is a sequence of nodes $(n_1, n_2 \ldots, n_m)$ such that there is an arc or line segment between two consecutive nodes n_i and n_{i+1} . The total cost for P is the sum of the costs of each individual arc along the path from the target node from the start node. The costs associated with each arc are represented as elements in a vector of positive values as in Equation 4.4 below, where $K(n_i, n_j)$ is denoted as the cost of traversing the $arc(n_i,\ n_j)$with each element of the vector representing a different cost variable, up the k_{th} cost variable (i.e. time, energy consumption, distance, etc) [43].

$K(n \quad)$ $,n$ $,n$ $,n$

The total cost vector of a path P is defined as the adding up of the all costs of all the arcs in P shown in Equation 4.5 below, where the total cost of the cost variable is denoted as $P)=\sum co$.

Weights can be added to each of the cost term to reflect the decision maker's priorities. For example, with appropriate cost weights, the path planner will locate a path which foregoes energy efficiency for reduced journey time or vice versa. When optimizing for a solution path P it is important to note that solution path P is said to be dominated, if there exists another solution path P',that improves at least one cost component of the cost vector K of *P* without making the other components poorer.

According to C. Barrett *et al.* [82] *Multicriteria optimisation* can also be described as "trade-offs of competing desirable qualities". These competing qualities can be illustrated on a curve called the *Pareto front* along which trade-offs of the criteria take place.

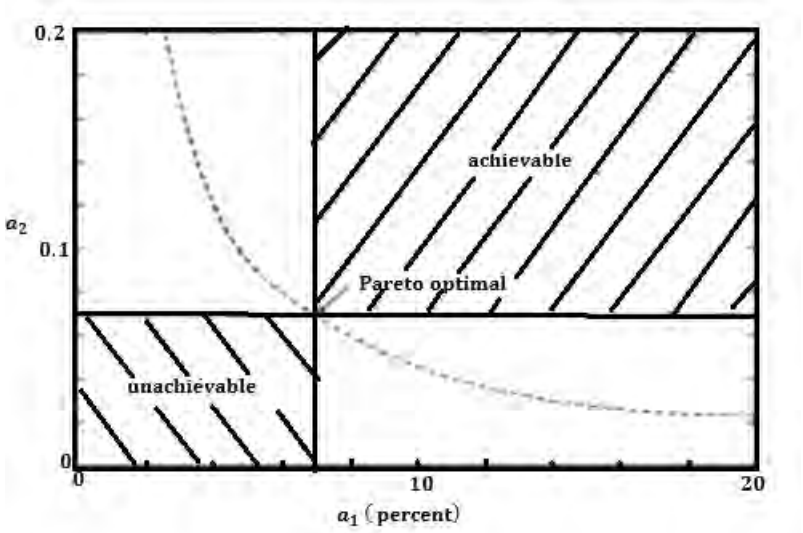

Figure 5.1: This figure shows a typical trade-off plot with a pareto front. The pareto front is the boundary which separates achievable and unachievable values. It is represented by the dashed curve shown in the figure (Adapted from [82]).

6 METHODOLOGY

In the previous Chapters the author discussed the relevant literature to the study. Firstly, the theoretical framework was introduced in chapter 1. Secondly, graph construction methods were introduced in chapter 2 explaining how a C-SPACE can be constructed from the real environment using techniques such as visibility graph method, the Voronoi graph method and the cell decomposition methods, one of which was used in the experiments conducted. Thirdly, graph search methods were then briefly discussed also in chapter 2 including the relevant Q-learning. Fourthly chapter 4 discussed how energy is consumed in a mobile robot and briefly introduces possible ideas to facilitate energy saving paths. Lastly related works are presented in chapter 5.

6.1 Conceptual design

6.1.1 Equipment and Software Packages

This is a software package including a MATLAB based software implementation of the Q-learning path planning algorithm and a graphical user interface with which to interact with the parameters of the algorithm. The software package was developed because it was essential to obtain particular simulation data. These include the output motion data (number of movements or steps) and the energy consumption output data of a mobile robot. This was essential in order to obtain the convergence behaviour of the algorithm and is the required outputs to be used in the energy consumption comparison of Q-learning and other algorithms. The version of Matlab used for the simulation is the 2012 one. The simulations were run on a Lenovo laptop with an Intel® Core™ i5-3210M processor. The flow chart of the Q-learning algorithm programmed in Matlab is shown below in Figure 6.1.

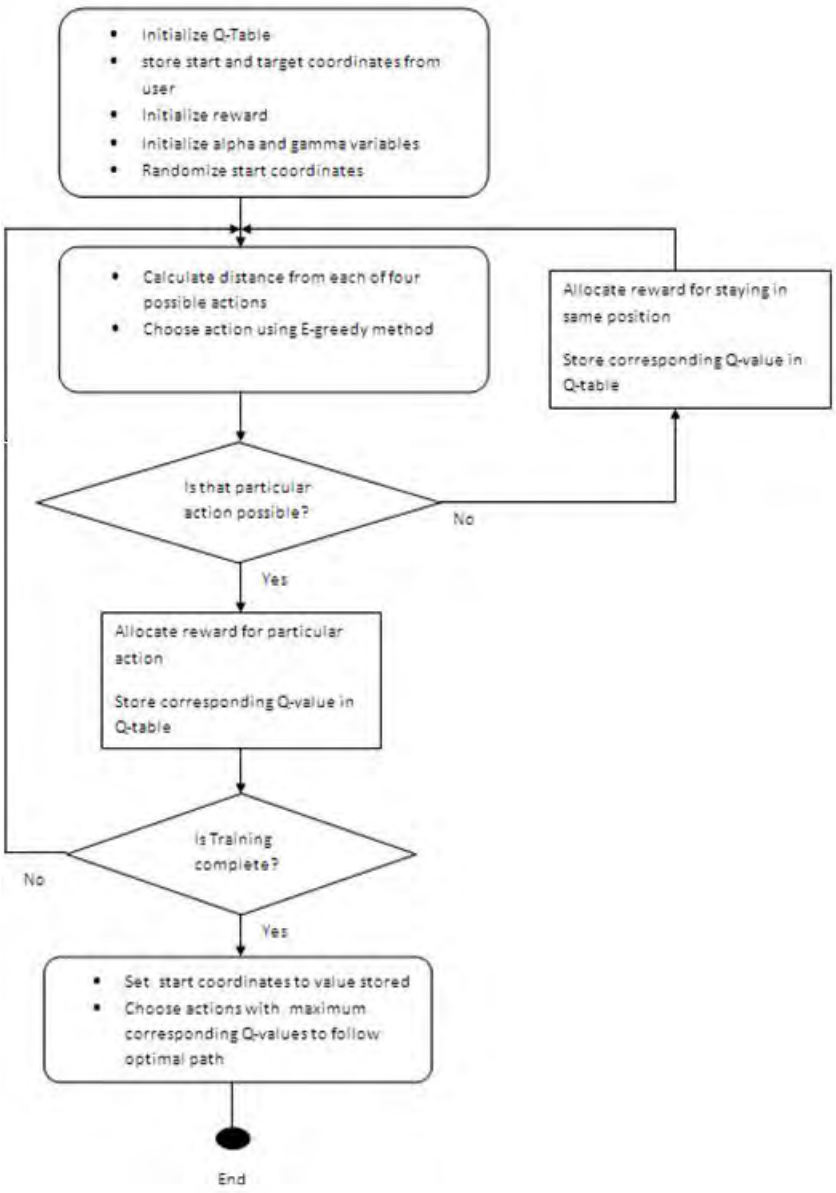

Figure 6.1: Flow chart of Matlab implementation of Q-learning algorithm

6.1.2 Motivation for using Matlab

The research involves developing a user friendly software version of the Q-learning path planner. The author found it was cheaper and easier to implement a fixed cell decomposition connectivity graph using Matlab, as Matlab allows matrix manipulation. Using the indexes of the different matrixes as nodes the author could change the values at particular row and column in the matrixes. Different values could represent different things such obstacles, start nodes and goal nodes. To produce movement the author could use methods to add or subtract indices' values, thus effectively changing the coordinates (location) of the robot on the grid map. Other programs did not offer the same flexibility as Matlab to manipulate matrices at low cost to the author.

6.1.3 Data collection & collation

In order to proceed with the experiments it is important to check that the Q-learning path planning algorithm works like it is supposed to. The first thing to check is that the algorithm learns a path from the start node to the target node. The plots that show that the algorithm learns are included in Appendix A1.

The next thing to check is that the algorithm converges. In order to do this it is essential to obtain results of average number of steps and energy used through a range of iteration values. These results will be displayed as tables and boxplots and will be shown in section 7.1 below. The last thing to check is that the algorithm allows exploration i.e. not deterministic. This last check was already confirmed as the algorithm relies on an e-greedy action select policy which introduces a certain amount of randomness to the action selection policy during navigation.

The next set of data collected is that pertaining to the comparison of the Q-learning algorithm against the A* algorithm, in terms of number of steps and energy consumption. This experiment will be conducted in both square and rectangular environments and will be shown in section 7.4.

6.2 Changes and their implications

The original Q-learning algorithm was programmed taking into account only 4 possible motions, i.e. up, down, left and right. This was found to work well in discovering the optimal paths, but was deemed to be wasteful in terms of energy consumption, because its paths had too many sharp turns, accelerations and decelerations. Examples of these paths shown are in Appendix A1 as a result the number of motions was changed from 4 to 8 motions i.e. algorithm was north, south, east, west, north-east, north-west, south-east and south-west. This change reduced the amount of unnecessary sharp turns, accelerations and decelerations.

6.2.1 Limitations

- The bigger the size of the occupancy grid i.e. the more nodes (states) in the connectivity graph available the more memory that is required as each action-state pair has to have a Q-value associated with it in order for paths to be generated. The memory needed increases exponentially as the size of the grid increases
- The algorithm takes longer to converge, the bigger the size of the map and the more obstacles on the map, as training stage is longer if many obstacles are present.
- If there is not sufficient training the algorithm will not form a path as some states were not explored.
- The action noise due to the exploration factor affects the output path length.
- The Q-learning algorithm is a slow algorithm and will require numerous iterations to converge to a good solution.

6.2.2 Proposed analysis

The first part of the proposed analysis on the data is to use the results and analyse the behaviours of this Q-learning path planner, in terms of convergence at different exploration factor and discount rate values. Next, the Q- learning path planning algorithm will be compared against the A* search algorithm in terms of number of steps and energy consumption to ascertain which performs better in environments of different obstacle densities. Lastly, the results of the algorithms will then be discussed in a Multi-Objective Optimisation framework because of the inherent conflict in optimising for the conflicting objectives of distance and energy.

6.3 Experimental Setup

The setup for the experiments is as follows. The robot construct used is that of a holonomic robot represented by a circle in two dimensions. This circular shape is based loosely on the *iRobot Create* mobile robot [82]. The robot's size is assumed to occupy about a third of a cell area. Also in realistic scenarios obstacles are not well defined and might not all take up the whole space of a square state.

The assumption for this experiment is that the obstacles are masses which occupy the centre of state squares. This means that the small circular mobile robots can transverse along the vertices of states which have obstacles into adjacent states, because the obstacles do not necessarily occupy the entire state. This construct was chosen because in a realistic model robot would not allowed to movement into these states and depending on the clustering of obstacles, as well as the area the robot takes up. So the robot might never reach its goal due to it been trapped. The difference to a realistic model of the robot and the environment, would be that the robot would just sense that an obstacle occupies that state and not transverse to an adjacent state via the vertex, but rather go around. This is ideal if the obstacle density of the environment is not so large.

The mobile robot in this experiment is able to move from a state (node) to any of its available neighbouring nodes. Any unavailable spaces in the environment C-SPACE will be classified as obstacles. The robot can move in any of the 8 direction if they are available. These directions are North, South, East, West, North- East, North-West, South-East and South-West respectively. The iRobot Create can move in all of these directions, since it can rotate around its centre moving to any predetermined angle as inputted by the user and then move in a straight direction. In fact the iRobot Create was used to measure the average energy consumed to make turns of 45, 90,135 and 180 degrees, both clockwise and anticlockwise. The iRobot Create has sensors which can measure the energy used in battery capacity (milliamp hours). Code was written which moved the robot through all of these angles, each 100 times. The energy values obtained were then averaged, after which they were then converted to the more familiar joules. Table 6.1 below shows the measure energy values for each angle in Joules.

The robot is assumed to be able to sense one unit in each of the 8 directions of motion. This is possible because of the circular shape of the robot. Sensors can be placed at angles of 45 degrees to each other. These sensors in a real scenarios would just be laser range finders or distance sensors, which should work pretty well of the obstacles in close. Hence, the one unit sensing distance used in these experiments make sense.

Smaller state spaces of 10 by 10 grid map were used for these experiments since this implementation of the Q-learning algorithm uses a lookup table. A larger state space will require impractically large amounts of memory. Larger state spaces generally need a form of generalisation; however, the convergence proof no longer holds [38].

6.3.1 Experiment setup one: Q-learning convergence test

In this experiment the Q-learning algorithm is run 20 times each at 10 different iteration values from 500 to a 5000, and the average number of steps is recorded. All the resultant number of steps averages at each iteration value is plotted on a graph to extract the convergence behaviour of the algorithm. This experiment produces the sets of results. The exploration factor ϵ_p is first set to 0.6; the discount rate is set to 0.1 and changed to 0.5 and 0.9 for first set. The discount rate γ and the exploration factor values are greater 0 to enable non-deterministic behaviour. The learning rate α is set to decrease gradually with each iteration value. The discounting rate is altered from 0.1 to 0.5 and 0.9, in order to see if there are changes in performance. In the next two sets the exploration factor ϵ_p is first set to 0.4 and 0.2 respectively, while learning rate and the discount factor change in the same way as in the first set.

6.3.2 Experimental setup two: Energy consumption comparison simulations

Firstly the Q-learning algorithm is compared with A* algorithm using simulations of a randomly generated square environments. The Q-learning path planner uses the graphical user interface to set the start and goal coordinates the obstacle density and the iteration value. The exploration factor ε_p and discount factor γ are set in the code to 0.8 and 0.9 respectively. The A* results are obtained using an A* Matlab demo developed by P. Premakumar(2010), which allows a user to manually position the robot's start and target locations are well as the obstacles[83]. The A* resultant paths were then adapted using some of the Q-learner code to look similar to the Q-learning resultant paths. This makes the results more comparable as they look similar.

The A* algorithm makes use of a path cost function which finds the neighbouring nodes/state in an open set with the minimum transversal cost. The total path cost is calculated by adding the path cost function to the heuristic. The A* algorithm also uses a path cost function plus the euclidean distance heuristic function to calculate the total cost of a path.

The first set of experiments is conducted in square environments. The obstacles are configured as follows: No obstacles, then 10% obstacle density and finally 20% obstacle density. An occupancy grid connectivity graph is used to represent the environments in the search space. The algorithms are all run from the same start and target point and can be compared in terms of the number of steps and energy consumption at all three earlier mentioned obstacle densities.

The second set of experiments are conducted in a different 20% obstacle density environment than mentioned above, with the difference being that the energy usage and number of steps is obtained from 20 different paths i.e. 20 different start and target coordinates whose results are averaged. In this setup the Q-learning graph search algorithm is compared with both the A* graph search algorithm and the least energy paths, for the 20 different start and target coordinates.

The third and fourth sets of experiments are conducted in rectangular shaped environments, and are repetitions of the first and second sets.

6.3.3 Multi-objective optimisation Analysis

In an attempt to optimise the Q-learning path planner the author uses both the distance and the energy consumption cost from an individual path used in simulation 4. A cost function for both the resultant Q-learning path and the A* path is computed, from the objective costs and then weights are applied to the path cost functions in order to optimise them. The resultant points based on the author's criteria are plotted as pareto curves in a multicriteria trade-off graph.

7 FINDINGS AND DISCUSSION

7.1 Q-learning convergence

In 1992 C. Watkins and P. Dayan presented a proof for Q-learning convergence. They proved that Q-learning will converge to an optimal policy in a discrete representation if certain conditions were met. The first condition is that the world (environment) has to be approximated by a Markov decision process as introduced earlier in section 2.3.2. The second condition was that the value of the learning rate α, where $0 \leq \alpha \leq 1$, has to decrease with (each succeeding action) each transition to a new state, such that each successive state has a reduced learning rate value than the previous state visited [10] [38]. This condition can be described in the form as shown below, where $n(x, a) = 1,2,3, \ldots$ is the number of times each individual state has been visited:

$$\alpha(x, a) = \frac{1}{n(x, a)}$$

$$= 1, \frac{1}{2}, \frac{1}{3}, \ldots$$

The third condition is that each state action pair (x, a) is visited an infinite number of times i.e. the environment and all possible actions are thoroughly explored [10] [38].The fourth condition is that a look-up table has to be used. Convergence is said not to be guaranteed if a lookup table is not used. If all the conditions are met the Q-learning algorithm has been proved to be convergent to an optimal policy with a probability of 1 [23]. It is important to note that if the environment is too large the use of the lookup table will result in impossibly large amounts of memory usage [38]. In a path planning problem this optimal policy is based on the criteria set in the path planning context i.e. the shortest distance path or energy efficient path.

The convergence of the Q-learning algorithm can be affected by an overestimation of the training stage action selection policy. In a non-deterministic MDP the action values obtained by the Q-learning algorithm can be interpreted as noisy samples of the actual action values

[51][77]. This noise gets updated by the value function of the algorithm. So the expected value of the maximum action value, used by the greedy navigation action-selection policy is an overestimation of the expected maximum value. The more stochastic the environment i.e. the more overestimation there will be inherently in the environment. This study has attempted to meet these conditions in the setup of the following experiments.

7.2 Q-learning convergence test results

Taking into account the three conditions mentioned in section 7.1, the Q-learning algorithm was run using the map in Figure 7.1 as the environment and three sets of results were produced. This section presents the first set of the results. The exploration factor $\varepsilon_p$is set to 0.6; the discount rate is set to 0.1 and changed to 0.5 and 0.9 for the second and third experiment respectively. The discount rate γ and the exploration factor values are greater than 0 to enable non-deterministic behaviour of the action-selection policy. The learning rate α is set to decrease gradually; with each iteration. The results are presented in table 7.1 followed by boxplots in Figures 7.1 and 7.2.

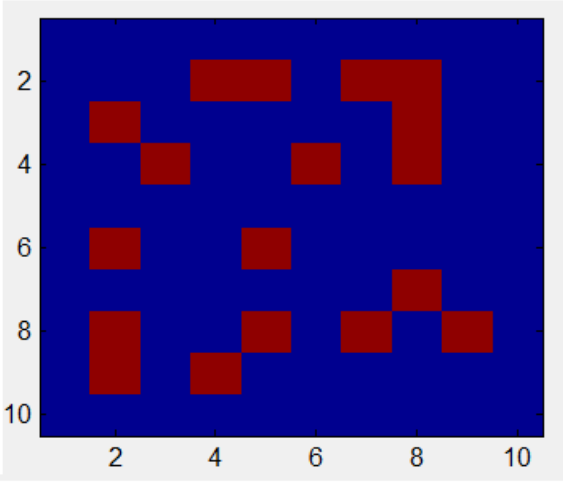

Figure 7.1: Q-learning convergence test map.
This figure shows Environment map used for the convergence test. Paths where mapped from start position (1, 10) to (10, 1) and run from 500 iterations to 5000 iterations.

Table 7.1: First set of Q-learning convergence results

Number of iterations	Average No. of steps	Average energy Usage (Joules)	Average Number of steps	Average energy Usage (Joules)	Average Number of steps	Average energy Usage (Joules)
	$\varepsilon_p = 0.6$ & $\gamma = 0.1$		$\varepsilon_p = 0.6$ & $\gamma = 0.5$		$\varepsilon_p = 0.6$ & $\gamma = 0.9$	
500	11.45	255.56	11.7	260.49	11.35	256.32
1000	11.45	252.96	11.6	265.24	11.35	252.19
1500	11.6	256.41	11.55	254.55	11.45	254.60
2000	11.3	251.89	11.35	250.97	11.55	254.73
2500	11.4	254.48	11.35	252.77	11.4	256.00
3000	11.45	256.73	11.35	252.06	11.15	249.71
3500	11.45	256.63	11.4	255.46	11.3	252.43
4000	11.35	253.15	11.35	253.15	11.35	252.61
4500	11.45	254.05	11.3	250.80	11.25	251.70
5000	11.45	254.05	11.2	250.43	11.55	258.72

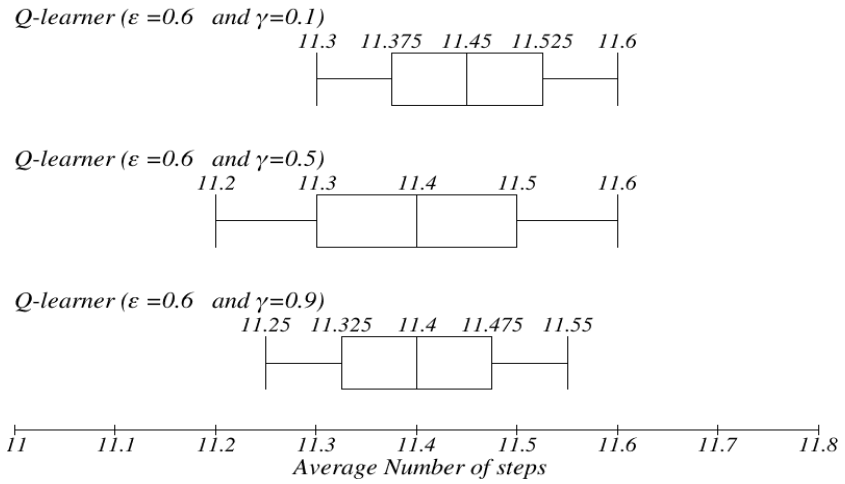

Figure 7.2: Boxplots of first set of Q-learning convergence results for the average number of steps

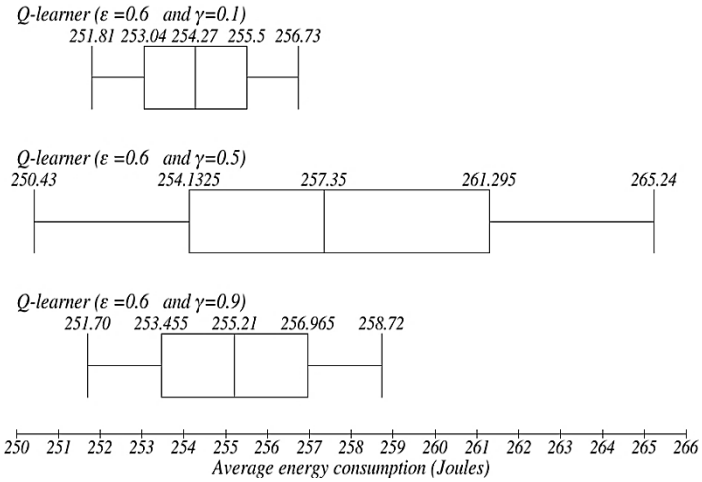

Figure 7.3: Boxplots of first set of Q-learning convergence results for the average energy consumption

Noticing the ranges of values i.e. the average number of steps and average energy consumption values in the table as well as the boxplots, it is clear to see that there is no apparent trend which is being followed. The values of average energy consumed do not approach definite values, as the discount factor gets increased. The values reveal no clear indication of convergence at iteration between 500 and 5000, which were the parameters set for this experiment.

The following section shows the second set of convergence results of the Q-learning graph search algorithm. The exploration factor ε_p is set to 0.4. The discount rate γ is set to 0.1 and changed to 0.5 and 0.9 for the second and third experiment. The discount rate γ and the exploration factor values are greater than 0 to enable non-deterministic behaviour of the action-selection policy. The learning rate α is set to decrease gradually; with each iteration. The results are presented in table 7.2 followed by the boxplots in Figures 7.3 and 7.4.

Table 7.2: Summary table 2 of Q-learning convergence results

Number of iterations	Average No. of steps	Average energy Usage (Joules)	Average Number of steps	Average energy Usage (Joules)	Average Number of steps	Average energy Usage (Joules)
	$\varepsilon_p = 0.4$ & $\gamma = 0.1$		$\varepsilon_p = 0.4$ & $\gamma = 0.5$		$\varepsilon_p = 0.4$ & $\gamma = 0.9$	
500	11.2	253.68	11	248.09	11	248.09
1000	11.05	248.81	11.1	249.53	11	248.09
1500	11.05	248.81	11.05	248.81	11	248.09
2000	11.1	248.09	11	248.09	11	248.09
2500	11	248.09	11	248.09	11	248.09
3000	11	248.09	11	248.09	11	248.09
3500	11	248.09	11	248.09	11	248.09
4000	11	248.09	11	248.09	11	248.09
4500	11	248.09	11	248.09	11	248.09
5000	11	248.09	11	248.09	11	248.09

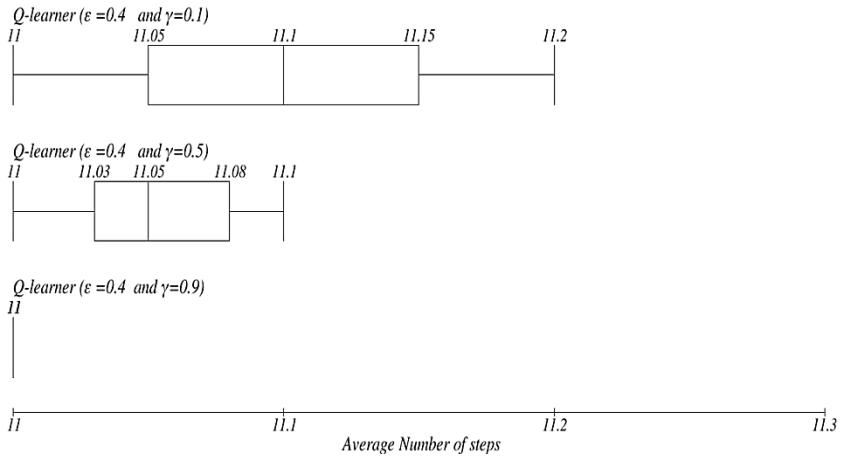

Figure 7.4: Boxplots of second set of Q-learning convergence results for the average number of steps

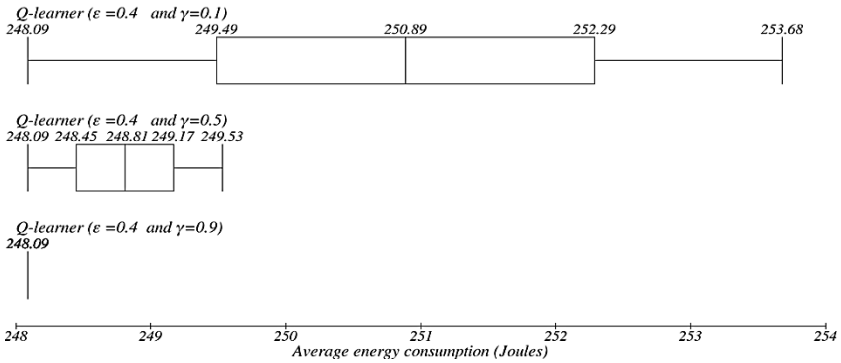

Figure 7.5: Boxplots of second set of Q-learning convergence results for the average energy consumption

In the second set of results there is a clear decreasing trend. The average number of steps approaches 11 and the average energy consumed approaches 248 joules, as the discount factor is increased. This indicates that overestimation is not as predominant in a lesser stochastic environment such as the one used in this experiment. This is explained as follows; a higher discount factor puts a heavier weighting on future rewards and invariably the Q(s, a) values. During the training phase the exploration action selection policy is stochastic with a 0.4 probability out of 1. It selects actions which would increase the value of future Q-values with a probability of 0.6 out of 1.When the training phase is completed the navigation action selection policy greedily chooses the actions in each consecutive state whose Q(s ,a) values are the maximum for that particular state, which is in line with Q-learning algorithm method [34][41].

The following section shows the third set of convergence results of the Q-learning graph search algorithm. The exploration factor ε_p is set to 0.2. The discount rate γ and the exploration factor values are greater than 0 to enable non-deterministic behaviour of the action-selection policy. The learning rate α is set to decrease gradually; with each iteration. The results are presented in table 7.3 below and the boxplots in the Figures 7.5 and 7.6.

Table 7.3: Summary table 3 of Q-learning convergence results

Number of iterations	Average No. of steps	Average energy Usage (Joules)	Average Number of steps	Average energy Usage (Joules)	Average Number of steps	Average energy Usage (Joules)
	$\varepsilon_p = 0.2$ & $\gamma = 0.1$		$\varepsilon_p = 0.2$ & $\gamma = 0.5$		$\varepsilon_p = 0.2$ & $\gamma = 0.9$	
500	11.05	249.15	11	248.09	11.05	249.15
1000	11	248.09	11	248.09	11	248.09
1500	11	248.09	11	248.09	11	248.09
2000	11	248.09	11	248.09	11	248.09
2500	11	248.09	11	248.09	11	248.09
3000	11	248.09	11	248.09	11	248.09
3500	11	248.09	11	248.09	11	248.09
4000	11	248.09	11	248.09	11	248.09
4500	11	248.09	11	248.09	11	248.09
5000	11	248.09	11	248.09	11	248.09

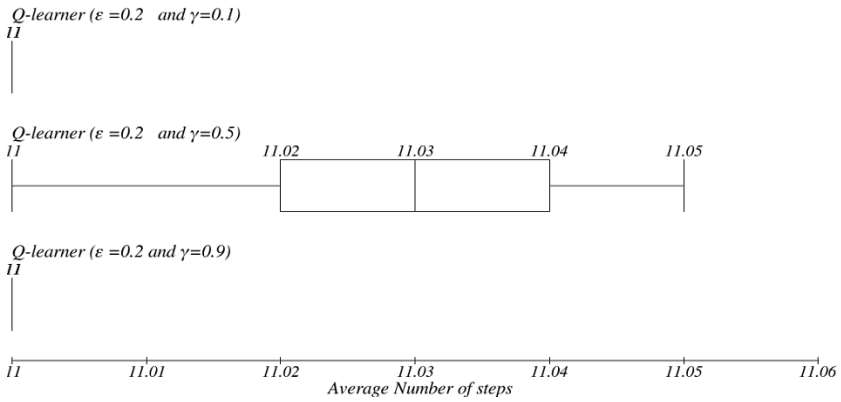

Figure 7.6: Boxplots of third set of Q-learning convergence results for the average number of steps

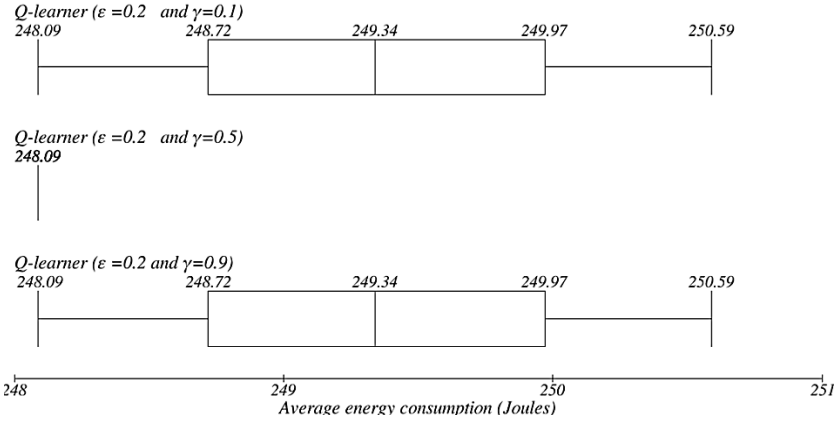

Figure 7.7: Boxplots of third set of Q-learning convergence results for the average energy consumption

In the third set of results it would seem like there is no clear decreasing trend. However, the range of the values which the path planner obtains is small. The small variation in the middle boxplots is that because overestimation does not altogether disappear; it just occurs less frequently. A high discount factor still plays an important role in putting a heavier weighting on future rewards, hence the smaller the range of values shown. The exploration factor ε_p is so small that at a probability of 0.2 out of 1, it is quite close to being deterministic, so it will converge despite the increased value of the discount rate γ, as deterministic MDP's converge very quickly.

7.3 Q-learning convergence test discussion

The results of the averages for the first set of results do not tend to any definite value i.e. converge. The reason for this result can be explained by the overestimation present in highly stochastic environment such as the one used in this experiment. In the first experiment an Exploration factor ε_p of 0.6 as used which make the environment within which learning takes place to be very non-deterministic in nature. The action selection policy of the Q-learning algorithm in the navigation stage chooses the actions with the maximum estimated Q-values. This view is shared by H. Hasselt (2010) and M. Azar *et al.* (2011)[79]. The following quote is taken directly from H. Hasselt (2010) work titled *Double q-learning, in which he* states:

> "In some stochastic environments the well-known reinforcement learning algorithm Q-learning performs very poorly. This poor performance is caused by large overestimations of action values. These overestimations result from a positive bias that is introduced because Q-learning uses the maximum action value as an approximation for the maximum expected action value"

This sentiment is echoed by M. Azar *et al.* (2011) who stated in in the paper titled *Speedy Q-Learning*. "This over-estimation is caused by a positive bias introduced by using the maximum action value as an approximation for the maximum expected action value"[79].

It to reason that if one were to decrease to the exploration factor then the overestimation would decrease as allowing the algorithm to converge which is the case when the exploration factor ε_p was set to 0.4 and 0.2 in experiments 2 and 3 respectively. It is important to note that though the algorithm converges towards a finite value, there might still be small overestimation present as long as the environment is stochastic to some degree as is the case in experiment 3, despite still tending towards convergence.

7.4 Search Graph algorithm comparisons

It is significant to note, as mentioned earlier in section 1.4 the algorithm of interest in this study is the Q-learning algorithm. However the A* is used as an exemplar by which to gauge the effectiveness of a Q-learning algorithm in saving energy. With this in mind it is necessary to question why the A* algorithm was chosen out of the several graph search algorithms available. It is of importance to note, that the terms “graph search” and path planner” are used interchangeably in this study and have the same connotation.

As introduced earlier in section 2.3, the most popular graph search algorithms are Dijkstra, BFS, DFS and A*.There are four features of a path planning algorithm that define performance, namely completeness, time complexity, space complexity and optimality. Completeness is the ability of the algorithm, to find a solution if one exists. Time complexity denotes how many computations are required to find a solution. Space complexity indicates how much memory is needed. Optimality is the measure of how good a solution is with respect to the path cost function. If a solution path is optimal it is denoted as admissible [52]. Table 7.4 below provides a brief comparison of the four algorithms with respect to the four features.

Table 7.4: Search Graph algorithm comparisons

	BFS	**DFS**	**Dijkstra**	**A***
Completeness	Guaranteed to reach a solution if one exists [52]	Not guaranteed to reach a solution if one exists [52]	Guaranteed to reach a solution if one exists [53]	Guaranteed to reach a solution if one exists [54]
Time complexity	Can be slow due to the exhaustive search of each branch from start node until finds solution. [52]	Finds solution without exploring much in a path hence the time is faster be less than the BFS graph search algorithm time [55]	Slower than A* because of Dijkstra might explore large area before finding target [56]	Very fast because it expands the fewest number of nodes , due to its heuristic function [53]
Space complexity	Requires amount of memory proportional to number of nodes [57]	Requires less memory as only needs to store nodes along the current path [52] [55]	Requires a lot of memory because it explore equally in each direction [56]	Requires substantial memory as saves entire open list of unexplored nodes [54]
Optimality	The solution paths admissible for small environments only [57]	The solution paths found may not be the best [52] [55]	The solution paths found are admissible if none of the edges have negative costs	The solution paths are admissible if the heuristic function is optimal [54]

7.4.1 Optimal paths comparison experiments in square environments

In the next section the Q-learning path planner is compared with the A* path planner. Three different square environments based on the cell decomposition method are used. The first is an environment with no obstacles, then the second an environment with 10% obstacle density and lastly an environment with 20% obstacle density. The Figures 7.7, 7.8 and 7.9 show the results of the comparison between the Q-learning algorithm and A* algorithm in terms of the average distance (in steps) travelled and the energy consumption in joules. It is important to note that the author is aware that the energy optimal paths normally contain less sharp turns. The purpose of this study is not to implement a new path finding algorithm which chooses paths with less turns, Rather it is to implement Q-learning in its original form then to compare it with another path finding algorithm to ascertain if it naturally uses less energy than the algorithm it is being compared with. An euclidean distance heuristic function is used with the A* algorithm because out of other available heuristic functions it is better suited for movements at any angle than the other heuristic functions. [58]

In The Q-learning path planner implementation the exploration factor ε_p is set to 0.8 and the discount factor γ is set to 0.9. The A* algorithm uses a path cost function which finds the neighbouring nodes/state in an open set with the minimum transversal cost. The total path cost is calculated by adding the path cost function to an heuristic. The A* algorithm also uses a path cost function plus the euclidean distance heuristic function to calculate the total cost of a path. A battery voltage of 14.4V was used to convert the energy consumption values from milliamp-hours to joules as it is battery that came with the iRobot mentioned above in section 7.2 above.

Smaller environments of 10 by 10 units were used in these first sets of experiments, because the Q-learning implementation in this study uses a look-up table. A larger state space will require impractically large amounts of memory. In large state spaces one will want to use some form of generalisation, but then the convergence proof no longer holds. The author chose the A* algorithm for comparison with the Q-learning, because it performed better on the average than the others.

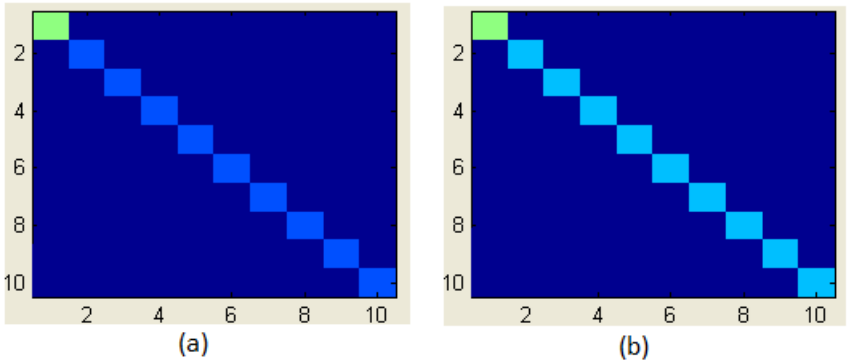

Figure 7.8: Navigating in a square environment with no obstacles (a) is the q-learning algorithm and (b) is the A* algorithm

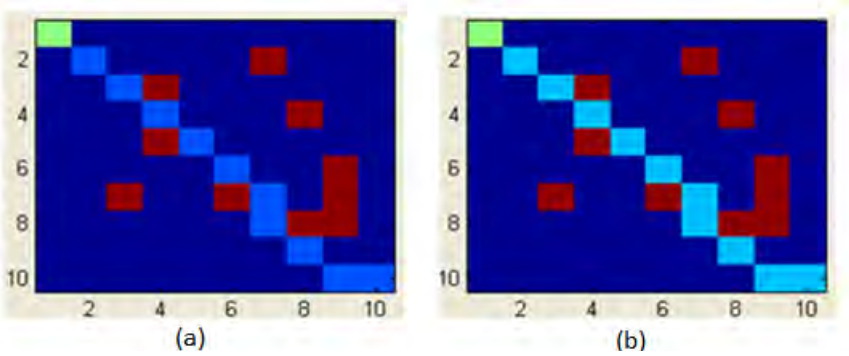

Figure 7.9: Navigating in a square environment with 10% obstacle density (a) is the q-learning algorithm and (b) is the A* algorithm

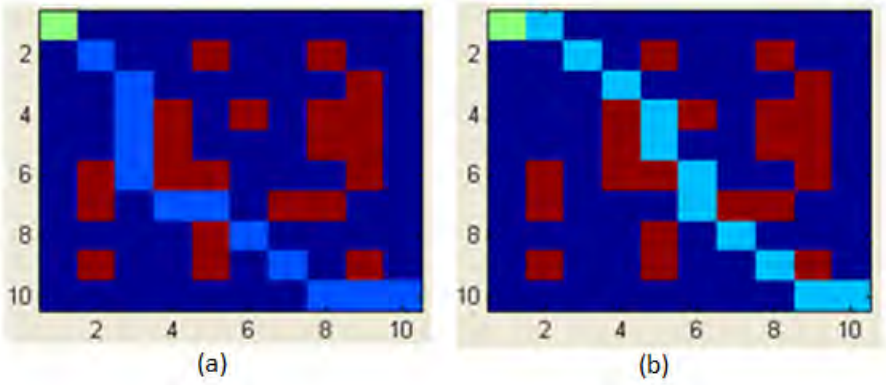

Figure 7.10: Navigating in a square environment with 20% obstacle density (a) is the q-learning path planner and (b) is the A* path planner

Table 7.5: Q-learning vs. A* in environments of different obstacle densities

	Q-learning	A*	Q-learning	A*	Q-learning	A*
	No obstacles		10% obstacle density		20% obstacle density	
Number of steps	9	9	10	10	12	11
Energy usage (Joules)	199.40	199.40	238.28	238.28	272.93	290.08

Table 7.5 gives a summary of the results of above Figures 7.7 to 7.9. The Q-learning path planner and the A* path finder obtained the same resultant paths for environments 0 to 10% obstacle density. In the 20% density environment however, the Q-learning path used consumes less energy (joules) than the A* path planner. This is due to the fact that in environments with large obstacle density, the non-deterministic nature of the Q-learning path planner affects the Q-values in its look-up table. A small exploration factor, affects the Q-table values allowing slightly less direct paths which save more energy.

7.4.2 Q-learning vs. A* multiple path comparison experiment

This next experiment involves a randomly generated square environment different from the ones used described earlier. It has an obstacle density of 20%. A comparison test of the Q-learning path planner, the A* path planner and the least energy paths are conducted, in terms of energy consumption and the number of steps from 20 different start and target coordinates. Figure 7.10 below shows the randomly generated environment. The results are shown below in table 7.6.

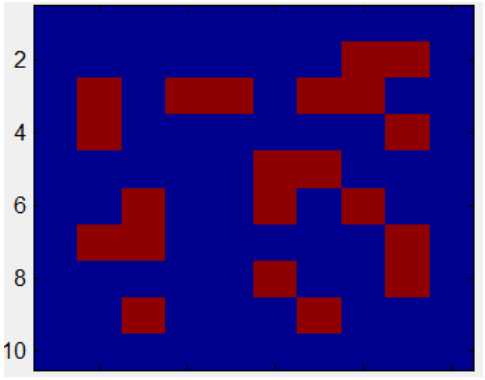

Figure 7.11: randomly generated 20% density square environment

Table 7.6: Multiple path comparison of Q-learning and A* algorithm

	Q-learning	A*	Least Energy paths
Distance in steps	9.21	8.95	9
Energy Consumption (joules)	215.14	221.88	202.69

While doing the comparison experiment it was discovered, that the Q-learning path planner used 3.04 % less energy than the A* path finder. The non-deterministic exploration nature of the Q-learning algorithm caused an over estimation of action values, affecting the Q-values in the lookout table. So although the Q-learning path planner converges; it converges to slightly less direct shortest paths due to the small exploration factor which in turn utilized less energy [51].

The A* path is on the average shorter than Q-learning because it expands/explores the fewest number of nodes due to, the optimal heuristic function (euclidean distance function) by not overestimating the actual cost function, This is because it navigates more directly to open nodes which have the least costs to transverse and hence uses more energy circumventing obstacles in the 20% obstacle dense environment. The author is unaware of any study in the area of path planning obtaining similar results. If other studies using the same approach come up with similar results then the findings should be worthy of closer consideration.

7.4.3 Optimal paths comparison experiments in rectangular environments

Industrial environments more often than not have rectangular shaped floors. So, it is essential to experiment in these types of environments too. Three different rectangular environments based on the cell decomposition method are used. The first is an environment with no obstacles. The second is an environment with 10% obstacle density. Third is an environment with 20% obstacle density. The Figures 7.7, 7.8 and 7.9 show the results of the comparison between the Q-learning algorithm and A* algorithm in terms of the average distance (in steps) travelled and the energy consumption in joules. Similarly as mentioned above the Q-learning path planner implementation using an exploration factor ε_p of 0.8 and a discount factor γ is set to 0.9. The A* algorithm also uses a path cost function plus the euclidean distance heuristic function to calculate the total cost of a path. A battery voltage of 14.4V was used to convert the energy consumption values from milliamp-hours to joules as it is battery that came with the iRobot mentioned above in section 7.2 above.

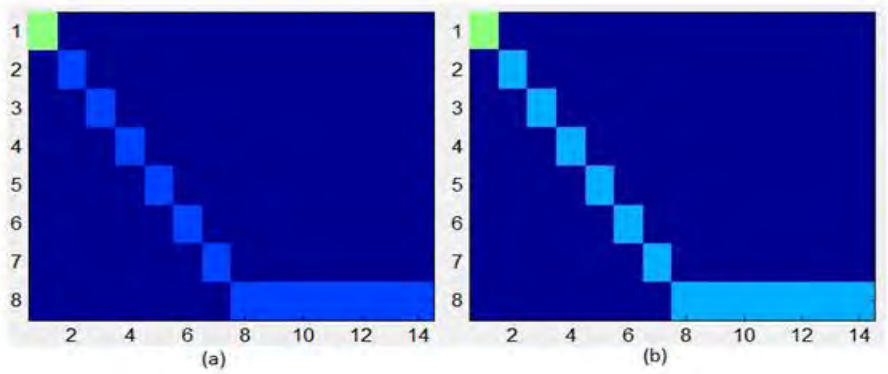

Figure 7.12: Navigating in a rectangular environment with no obstacles (a) is the q-learning algorithm and (b) is the A* algorithm

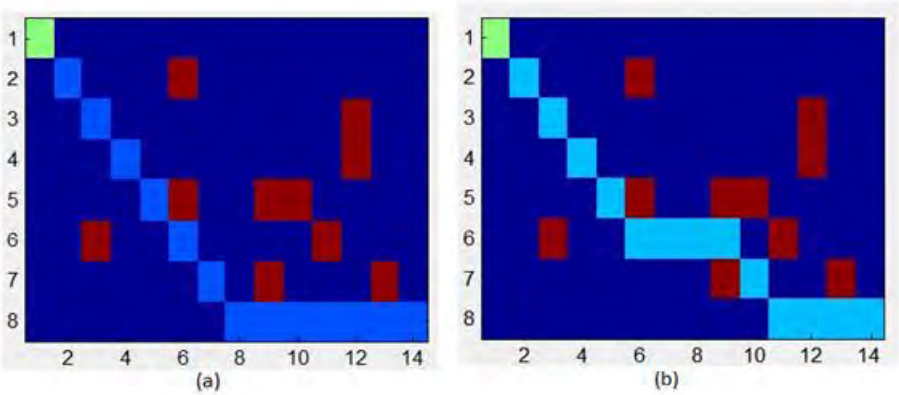

Figure 7.13: Navigating in a rectangular environment with 20% obstacle density (a) is the Q-learning algorithm and (b) is the A* algorithm

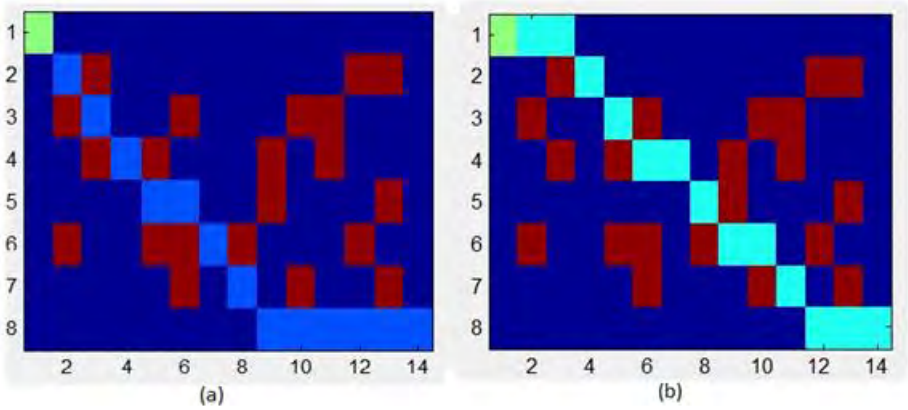

Figure 7.14: Navigating in a rectangular environment with 20% obstacle density (a) is the Q-learning algorithm and (b) is the A* algorithm

Table 7.7: Q-learning vs. A* performance in rectangular environments

Path planners	Q-learning	A*	Q-learning	A*	Q-learning	A*
	No obstacles		10% obstacle density		20% obstacle density	
Number of steps	13	13	13	13	13	13
Energy usage (Joules)	269.34	269.34	269.34	300.96	294.22	322.2167

Table 7.6 gives a summary of the results of the above Figures 7.11 to 7.13. It is clear to see that the Q-learning path planner and the A* path finder obtained the same number of steps for environments of 0 to 20% obstacle density. However, as the obstacle density was changed to 10% and 20% respectively the Q-learning path planner consumed less energy (joules) than the A* path planner. In terms of the number of steps it seems that in this case the obstacles did not cause major detours. As a result the path planners still obtained the same length of shortest paths. Major detours for example could have been c or u-shaped obstacle configurations. However the merits of this finding are that environments of varying complexities will not pose a significant problem for the Q-learning path planner. As mentioned above in section 6.3, obstacles are not well defined and might not all take up the whole space of a square state, which allows the robot to navigate along the vertices between two obstacles as seen in Figure. 7.13

In terms of energy usage, the results are similar to the results obtained for the square environments mentioned before in section 7.2. In the rectangular environments with large obstacle density, the non-deterministic nature of the Q-learning path planner affects the Q-

values in its lookup table. A smaller exploration factor, affects the Q-table values allowing slightly less direct paths which save more energy by making less wasteful turns.

7.4.4 Q-learning vs. A* multiple path comparison experiment II

This is essentially the same experiment as in 7.2.2, but the environment used is rectangular in shape. It has an obstacle density of 20%. A comparison test of the Q-learning path planner, the A* path planner and the least energy paths are conducted, in terms of energy consumption and the number of steps from 20 different start and target coordinates. Figure 7.14 below shows the randomly generated environment. The results are also shown in table7.2 below

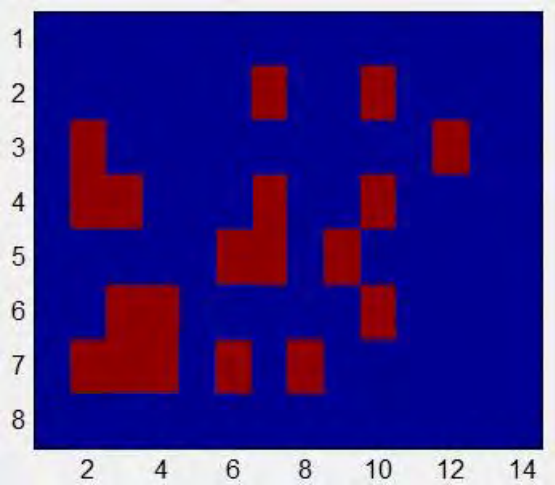

Figure 7.15: randomly generated 20% density rectangular environment

Table 7.8: Multiple path comparison of Q-learning and A*

	Q-learning	A*	Least Energy paths
Average Distance in steps	9.6	9.6	9.6
Energy Consumed in joules	218.03	221.69	210.93

While conducting the second comparison experiment it was discovered, that the Q-learning path planner used 1.68 % less energy than the A* path finder. The non-deterministic exploration nature of the Q-learning algorithm causes an over estimation of action values, affecting the Q-values in the lookout table. Although the Q-learning path planner converges, it converges to slightly less direct shortest paths. This is probably due to the small exploration factor, which in turn utilizes less energy [51]. The difference between the energy consumed by the Q-learner and the A* in rectangular environments is less noticeable than the square environments. This is not unrelated to the reduced options of the paths, a result of lack of symmetric of the environments. This is made more obvious when there are obstacles within the environment. For the same reason single optimal paths are more likely to be found. This clarifies why the shortest paths are the same length.

7.4.5 Q-learning vs. A* discussion of comparison experiments I and II

If one were to pay careful attention to the results in table 7.5 it would be clear to see that compared to the least energy paths the Q-learning algorithm uses 5.79% more energy than the least energy paths. In terms of square environments the Q-learning algorithm in its original form has not been effective as an energy-saving device. However, the Q-learning path planning algorithm uses 3.04% less energy than the A* path algorithm, so it can be deem more effective in obtaining energy saving paths than the A* algorithm.

In the case of rectangular environments, if one were to pay careful attention to the results in table 7.7 it would be clear that compared to the least energy paths the Q-learning algorithm uses 3.26 % more energy than the least energy paths. So in terms of rectangular environments the Q-learning algorithm in its original form generating energy saving paths, one can motivate that the difference in effectiveness is less pronounced although the least energy paths are still substantial better. The Q-learning path planning algorithm uses 1.68% less energy, than the A* path algorithm, so it can still be deem fractionally more effective in obtaining energy saving paths than the A* algorithm.

From table 7.4 one can motivate that the A* algorithm on the average performs better than the other earlier mentioned graph search algorithms. Therefore, it should be so it will be of interest to evaluate the energy consumption of the other algorithms mentioned earlier in a simulation setup similar to the one used in this study. This is to enable one to obtain a more complete picture of the effectiveness of the Q-learning algorithm.

7.5 Multi-criteria analysis

Using the same 20% density square environment as described above in section 7.22 and a single path from the start coordinate (10,1) to the target coordinate (1,10) (see Figure 7.15 below), can be optimised.

Table 7.9: single path results from the start coordinate (10,1) to the target coordinate (1,10)

Path planners	Q-learning	A*
Number of steps	11	11
Energy consumption (Joules)	262.6	265.7

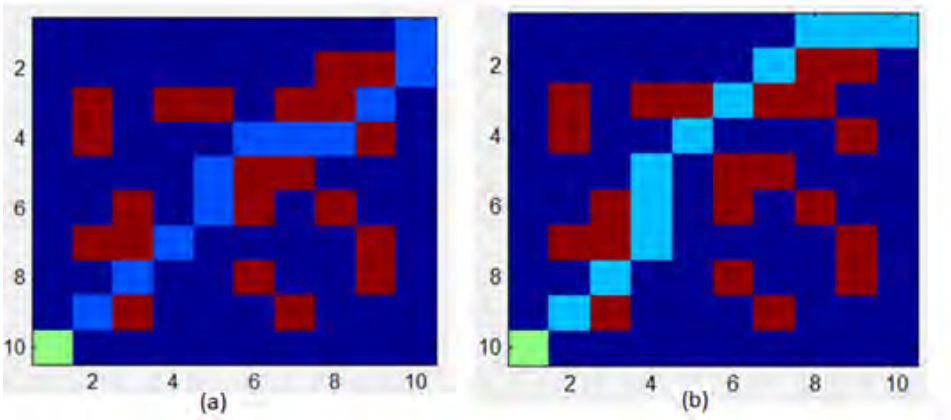

Figure 7.16: The Paths from the start coordinate (10, 1) to the target coordinate (1, 10) (a) is the Q-learning path from (10,1) to (1,10) and (b) is the Q-learning path From (10,1) to (1,10).

According to the multicriteria framework in section 4, objectives are considered in the form of goals which are expressed as inequalities like those below [43]. To obtain the constraints, certain steps are needed. Firstly, the euclidean distance $Euclid_{dist}$ is measured from the start to the target, in terms of steps. It is impossible to traverse to the target from the start in fewer steps than the euclidean distance. Similarly, if the minimum energy consumption rate $Emin_{Rate}$ in $^{joules}/_{step}$, then at least $Euclid_{dist} \times {}^{Emin}/_{step}$ is needed to reach the destination such that:

$Euclid_{dist} = 9$ *steps*

$Emin_{Rate} = 24.92$ *joules per step*

$Emin_{Total} = 9 \times 24.92 = 224.28$ *joules*

Secondly aspiration values of 120% of these optimistic values are used to obtain the goal constraints as shown below [43].

Energy ≤ 269.2 *Joules (rounded up to 270)*

Distance $\leq$ *10.8 Steps (rounded up to 11)*

The cost functions for the Q-learning path planner and the A*path planner using the result values from table 6 above are as follows:

$J_Q = 262_Q + 11_Q$

$J_{A*} = 265.7_{A*} + 11_{A*}$

Table 7.10: single path results, from the start coordinate (10,1) to the target coordinate (1,10)

W_E (%)	W_E (%)	J_Q	J_{A*}
100	0	$J_Q = 262_E + 0_Q$	$J_{A*} = 265.7_E + 0_Q$
90	10	$J_Q = 235.8_Q + 1.1_Q$	$J_{A*} = 239.13_{A*} + 1.1_{A*}$
80	20	$J_Q = 209.6 + 2.2_Q$	$J_{A*} = 212.56_{A*} + 2.2_{A*}$
70	30	$J_Q = 183.4_E + 3.3_D$	$J_{A*} = 185.99_E + 3.3_D$
60	40	$J_Q = 157.2_Q + 4.4_Q$	$J_{A*} = 159.42_{A*} + 4.4_{A*}$
50	50	$J_Q = 131.35_Q + 5.5_Q$	$J_{A*} = 132.85_{A*} + 5.5_{A*}$
40	60	$J_Q = 105.08_Q + 6.6_Q$	$J_{A*} = 106.28_{A*} + 6.6_{A*}$
30	70	$J_Q = 78.6_Q + 7.7_Q$	$J_{A*} = 79.71_{A*} + 7.7_{A*}$
20	80	$J_Q = 52.4_Q + 8.8_Q$	$J_{A*} = 53.14_{A*} + 8.8_{A*}$
10	90	$J_Q = 26.2_Q + 9.9_Q$	$J_{A*} = 26.57_{A*} + 9.9_{A*}$
0	100	$J_Q = 0_Q + 11_Q$	$J_{A*} = 0_{A*} + 11_{A*}$

Lastly the author then optimised the cost functions for different ratios of energy optimality vs. distance optimality, using weights W_E and W_D. The corresponding pareto curves is shown below in Figure 7.16.

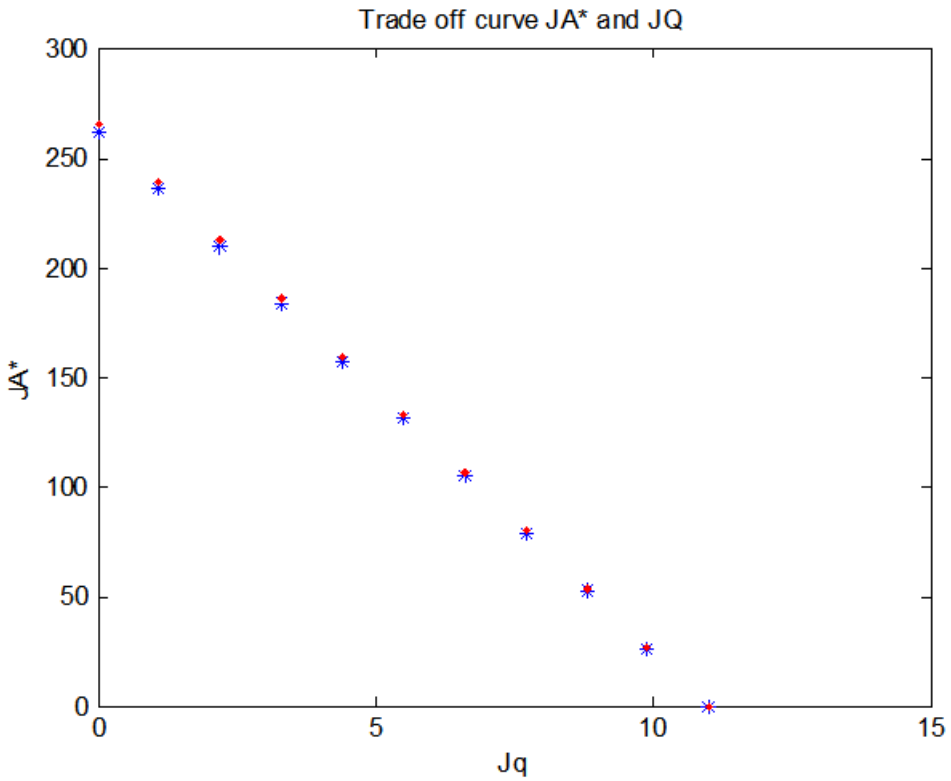

Figure 7.17: This figure is shows a trade-off plot with the pareto curves of Q-learning optimal path and A* optimal path. The red dots represent the Q-learning pareto curve and the A* path pareto curve is represented by the blue asterisks. The red dots and blue asterisks are of no significance but to distinguish the pareto curves**.**

7.6 Final Comments

The merit of the study is that it gives a platform for future studies to profit from because there is not much literature or results on the energy saving application of the Q-learning algorithm as a path planner. The implications of the findings for future studies in the area seem obvious. First the Q-learning algorithm is not specific to the area of path planning, However, A* algorithm and the others mentioned earlier are typical path planning algorithms. Secondly this study points to the value of using learning algorithm research and implementations in comparison with existing solutions. This is to provide insights in the area. It is also a way to improve on the existing solutions to contemporary research problems.

Energy savings is a topic on the high agenda of many application systems, not least path-planner systems. Therefore, any further insights emanating from research studies in the area are likely to attract keen interests from industries and other stakeholders. For the same reason, the judges of the Eta Awards (an annual event sponsored by Eskom) specifically focused in 2004 on energy savings as opposed to previous years where the main focus was on innovative endeavours. According to Steve Lennon, chairperson of Eta Steering Awards for that year, “Innovation, although important, is not the main criterion; but rather improvements in terms of affordability…The Eta Awards strive to recognise and reward innovative thinking and practical application of efficient energy usage.” [59]

8 CONCLUSIONS

This chapter summarises the thesis and discusses the research limitations of the study. Recommendations for future work using the Q-learning approach will also be discussed.

8.4 Summary

This thesis has implemented the Q-learning algorithm to evaluate the energy usage of its generated paths, within unknown and unstructured environments, which is the main contribution of this dissertation. The methodology uses the cell decomposition graph construction search space to obtain optimal paths for mobile robots. The method makes use of a non-deterministic MDP to explore the environment, gain the individual value of all the actions in the available state space, greedily implementing the actions with the highest values in order to obtain the optimal path. Compared with the A* graph search algorithm in [53] [56], the simulation results in section 7.4 show that the Q-learning graph search algorithm uses less energy in dense unknown and unstructured environments with static obstacles. This is crucial for the contemporary applications of path planning.

Though the Q-learning implementation on average uses more energy than the least energy path in section 7.3.2, it is a step in the right direction to achieve the goal of successfully implementing algorithms which produce less energy paths. Though the implementation of this algorithm is not entirely novel, it illuminates the concept of using learning algorithms to improve the energy efficiency of mobile robots, which may essential for sustainable mobile robot applications in the near future. The algorithm is simpler and easier to implement than the other earlier mentioned graph search methods, for use in unknown and unstructured environments, and can be used in applications such as, dangerous situation navigation, ocean exploration and space exploration [3].

8.5 Study limitations

To a great extent, this study has succeeded in implementing the Q-learning algorithm in the form of a path planner in unknown and unstructured environments, with a cell decomposition structure. Nevertheless, there are some limitations worthy of consideration in future studies in the area. The limitations of the study are as follows:

In this study the mobile robot took a holonomic construct, represented as a small circular shape in two dimensions. Each grid square will represent a state in square environments and a grid rectangle will represent a state in rectangular environments. In this implementation the robot can move along vertices of the grid squares in the occupancy grid. The obstacles will be assumed to be more centrally positioned in the states. This means that the robot will not be able to navigate through the middle of states that have obstacles. The obstacles will still be able to navigate along the vertices of the obstacle states to enter a new state.

It assumed that the robot can sense one state in all 8 directions of movement and also sense one unit in all 8 directions. For a more realistic model individual sensor ranges can be measured and implemented, but should be of little significance as one unit can considered as close enough to reduce error significantly.

8.6 Recommendations for future work

Though the Q-learning algorithm obtains better results than the popular A* algorithm in terms of energy consumption within a randomly generated (unstructured) environment, there is still some work recommended to be done in the future. The Q-learning algorithm should be tested on a more intricate and realistic model of the environment. The environments used in the study are constructed as two–dimensional configuration spaces, where the obstacles are represented as coloured squares in a grid. A more realistic construction requires a three-dimensional model of the environment, while still keeping the unstructured nature of said environment. Obtaining more accurate individual sensor ranges as mentioned earlier in section 8.2 is also worthy of attention.

The exploration/research into the use of the Q-learning path planning algorithm on more suitable mobile robot simulators which produce more practical environments than those constructed in Matlab is needed. The Matlab implementation merely uses matrix manipulation to exhibit the capability of the algorithm rather than provide a practical testing environment for the Q-learning algorithm.

- Implementing the Q-algorithm path planner on a real mobile robot and then comparing results obtained with the results from the simulations in order to realise the effectiveness of the simulation implementation in promoting real life energy savings.

9 BIBLOGRAPHY

[1] P. Bright. “How Are Robots Used in Industry?” Internet: www.ehow.com/how-does_4572605_how-robots-used-industry.html, Oct. 28, 2008, [Nov. 7, 2013].

[2] J. Jobin, “Industrial Robots: 5 most popular applications.” Internet: blog.robotiq.com/bid/52886/Industrial-robots-5-most-popular-applications, Feb. 9, 2012.

[3] B. Brumson, “*New Applications for Mobile Robots”* Internet:http://www.robotics.org/content-detail.cfm/Industrial-Robotics-Feature-Article/New-Applications-for-Mobile-Robots/content_id/3362, April 2012.

[4] Fraunhofer, “Automatic Path Planning*,”* Internet:http://www.fcc.chalmers.se/geo/profile/automatic-path-planning, 2013.

[5] E. Dijkstra, *A note on two problems in connection with graphs, Numerische Mathematik,* vol. 1, 1959, pp. 269-271.

[6] N. Buniyamin and N. Sariff. “An Overview of Autonomous Mobile Robot Path Planning Algorithms,” presented at the *4th Student conference of research and development*, 2006, pp. 183-188.

[7] J. Latombe. *Robot Motion Planning*. Boston: Kluwer Academic Publishers, 1991, pp. 1-6.

[8] R. Siegwart, I. Nourbakisk and D. Scaramuzza*. Introduction to Autonomous Mobile Robots, 2nd ed*. R. Arkin, Ed. Cambridge, Massachusetts: The MIT press, 2004, pp. 288-291 & 371-393.

[9] O. Khatib and B. Siciliano, *Springer Handbook of Robotics*. Berlin: Springer Science+Business Media, 2008, pp. 109 – 128 & pp. 827 – 850.

[10] C. Watkins and P. Dayan, *Technical Note: Q-learning,* Boston: Kluwer Academic Publishers, 1992, pp.1-2.

[11] Dictionary.com, “autonomous,” in Dictionary.com Unadridged.Source location: Random House Inc, Available: http://dictionary.reference.com Accessed: May 13, 2014.

[12] All Psych and Heffner Media Group, Inc. "Reinforcement." Available: http://allpsych.com/psychology101/reinforcement.html.

[13] P. Dayan and C. Watkins, *Encyclopedia of Cognitive Science*, London: Department of Computer Science,University of London, 2002, p. 2

[14] W. Burgard and C.Stachniss, *The Markov Decision Problem:Value Iteration and Policy Iteration,* Freiburg: University of Freiburg, 2003, pp.1-43.

[15] P. Duygulu, CS 461 Chapter 2, Topic: “Intelligent Agents”*,* Bilkent University, Ankara, 2008.

[16] http://wordnetweb.princeton.edu/perl/webwn?s=Localization [Online].

[17] T. Lozano-Perez, "A Simple Motion Planning Algorithm for General Purpose Manipulators," *IEEE Journal Of Robotics And Automation,* vol. no. 3, pp. 224-238, June. 1987.

[18] H. Miao, "Robot Path planning in Dynamic Environments using a Simulated Annealing Based Approach,” M.S. thesis, Queensland University of Technology,Queensland, 2009.

[19] R. Muhammed, “Computational Geometry.” [Online]. Internet: www.personal.kent.edu/~rmuhamma/Compgeometry/MyCG/CG-Applets/VoroDiagram/vorocli.htm.

[20] E. Roberts, "Motion Planning in Robotics," Internet: http://www-cs-faculty.stanford.edu/~eroberts/courses/soco/projects/1998-99/robotics/basicmotion.html. [11 July 2012].

[21] P. Norvig and S. Russel, *Artificial Intelligenc: a modern approach*, Prentice Hall, 1995, p. 82.

[22] Z. Zhongli and W. Qiang, "Reinforcement Learning Model,Algorithms and its Application," presented at *the 2011 International Conference on Mechatronic Science,* Electric Engineering and Computer, Jilin, 2011.

[23] L. Kaebling, M. Littman and A. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*,vol no. 4, pp. 237-285, May 1996.

[24] A. Barto and R. Sutton, "Reinforcement Learning," *Journal of Cognitive Neuroscience,* vol. 11, pp. 126–134,[Jan 1999].

[25] M. Sewell, Machine Learning, London: Department of Computer Science, 2007, pp. 1-3.

[26] B. Sendhoof and Y. Jin, "Pareto-Based Multiobjective Machine Learning:An Overview and Case Studies," in *IEEE Transactions On Systems, Man, And Cybernetics*, 2008, pp. 397-415

[27] L. HaiHua and W. Weishan, "Machine Learning Applications in Rough Set Theory," in *Internet Technology and Applications*, 2010, pp. 1-3.

[28] R. Simmons and S. Koenig, "Unsupervised learning of probabilistic models for robot navigation," in *IEEE International Conference on Robotics and Automation*, 2006, pp. 2301 - 2308

[29] P. Dayan, M. Sahani and G. Deback, *Unsupervised Learning,* Massachusetts: MIT Encyclopedia of Cognitive sciences, 1999, p. 1.

[30] J. Zaragoza and E. Morales, "An Introduction to reinforcement learning," in *Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions*, IGI Global, 2012, pp. 63-67.

[31] C. Gaskett, "Q-Learning for Robot Control,". Ph.d. Thesis, Supervisor A. Zelinsky, Department of System Engineering.The Australian National University, Australia, 2002.

[32] M. Puterman, Markov Decision Processes: Discrete stochastic dynamic programming, New York: Wiley-Interscience Publication, 1994.

[33] B. Huang, G. Cao and M.Guo, "Reinforcement Learning Neural Network To The Problem Of Autonomous Mobile Robot Obstacle Avoidance," in *Proceedings of the Fourth International Conference on Machine Learning and Cybernetics*, 2005, pp. 85-89

[34] W. Hsu. CIS 732. Class Lecture, Topic. Introduction to Reinforcement Learning: Q Learning, Department of Computing and Information Sciences, Kansas State University, Kansas state, Oct. 19, 2002.

[35] K. Maček, I. Petrović and N. Perić, "A reinforcement learning approach to obstacle avoidance of mobile robots," in *7th International workshop on advanced motion control* ,2002, pp. 462-466.

[36] C. Sims, "Reinforcement Learning: Model-based," University of Rochester,Rochester, New York ,July 24. 2012.

[37] P. Dayan, "Interaction between Model-Free and Model-Based Reinforcement Learning"*,Youtube.* Online. Available: http://www.youtube.com/watch?v=1z6agTRvOo [July, 2013].

[38] M. Humphreys, "Action Selection methods using reinforcement learning," PhD, Cambridge, 1997.

[39] Z. Hamsah, "Are we learning now/" Online. Available: http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol2/zah/article2.html [Nov. 3, 2013]

[40] X. Fern, "Reinforcement Learning cont." Class notes for CS434, College of Engineering, Oregon State University,Nov. 5, 2012. [Online]. Available: http://classes.engr.oregonstate.edu/eecs/fall2012/cs434/notes/RL-2.pdf.

[41] T. Eden, A. Knittel and R. van Uffelen,Tutorial on reinforcement learning [Online]. Available: http://www.cse.unsw.edu.au/~cs9417ml/RL1/index.html, 2013.

[42] M. Shahab, "Energy-Efficient Motion Control of Mobile Robots, EE 656, " King Fahd University of Petroleum & Minerals, 2009.

[43] J. Fernandez, L. Gonzalez,L. Mandow and J.PeÂrez-de-la-Cruz, "Mobile robot path planning: a multicriteria approach," *Engineering Applications of Artificial Intelligence,* vol. 12, pp. 543-554, 1999.

[44] L. Zamstein, A. Arroyo and S. Keen, "Path planning using reinforcement learning on a real robot platform," in *Conference on Recent Advances in Robotics*, 2006, pp. 1-4.

[45] C. Li, J. Zhang and Y. Li, "Application of Artificial Neural Network Based," in *Proceedings of the 2006 IEEE International Conference on Information Acquisition*, 2006, pp. 978-982.

[46] Y. Li, C. Li and Z. Zhang, "Q-Learning Based Method of Adaptive Path Planning," in *Proceedings of the 2006 IEEE International Conference on Information Acquisition*, 2006, pp. 984 - 987.

[47] I. Goswami, P. Das, A. Konar and R. Janarthanan, "Conditional Q-learning Algorithm for Path-Planning a Mobile Robot," in *International Conference on Industrial Electronics, Control and Robotics*, 2010, pp. 24 – 27.

[48] D. Tamilselvi, S. Mercy Shalinie and G. Nirmala, "Q Learning for Mobile Robot Navigation in Indoor Environment," in *IEEE-International Conference on Recent Trends in Information Technology*, 2011, pp. 324 -329.

[49] Y. Mei, Y. Lu, Y. Hu and C. Lee, "Energy-Efficient Motion planning for mobile robots," in *Proceedings ICRA '04. 2004 IEEE International Conference*, 2004, pp. 4344-4349.

[50] M.Dupac and C.Popirlan, "An Optimal Path algorithm for Autonomous Search Robots," *The Annuals of the University of Croaiova,* vol. 36, no. 1,pp. 36-47, 2009.

[51] H. Hasselt, “Insights In Reinforcement Learning:formal analysis and empirical evaluation of temporal difference learning,” Ph.D thesis, Organisation for Scientific Research(NWO), Utrecht: Netherlands 2011.

[52] “Class notes on search,” Class Notes for CSci 4511w, Department of Computer Science and Engineering,University of Minnesota,2010 [Online] Available: http://wwwusers.cs.umn.edu/~gini/4511/search.html

[53] A. Patel, "Introduction to A*," May 2004.[Online]. Available: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparision.html.

[Accessed:13 Sep 2013].

[54] Robin, “A Star algorithm”, intelligent.worldof computing.net, Dec. 2009, [Online]. Available: http://intelligence.worldofcomputing.net/ai-search/a-star-algorithm.html,"

[Dec. 12 2013].

[55] Robin, "Depth first Search," intelligent.worldof computing.net, Dec. 2009 [Online]. Available: http://intelligence.worldofcomputing.net/ai-search/depth-first-search.html. [Dec. 12 2013].

[56] P. Lester, "A* Pathfinding for beginners," [Online]. Available: http://www.policyalmanac.org/games/aStarTutorial eng.htm. [Accessed 2005].

[57] Robin, “Breath first Search," intelligent.worldof computing.net, Dec. 2009 [Online]. Available: http://intelligence.worldofcomputing.net/ai-search/breadth-first-search.html, [Dec. 12 2013].

[58] E. W. Dijkstra “A Note on Two Problems in Connexion with Graphs,” Numerische Mathematik, Vol. l, No. 1, 1959, pp. 269-271.

[59] T. McDonald, *Eta Awards 2004 focus on energy savings,* Energy Management News, 2007, p. 7.

[60] M. Kurant, A. Markopoulou and P. Thiran, "On the bias of BFS (Breadth First Search)," in *International Teletraffic Congress (ITC 22)*, 2010, pp. 1-8.

[61] Z.Zhongli. and W. Qiang "Reinforcement Learning Model,Algorithms and its Application," in *International Conference on Mechatronic Science, Electric Engineering and Computer*, 2011, pp. 1143-1145.

[62] G. Hayes, "*Reinforcement Learning",* School of Informatics,University of Edinburgh,Edinburgh, Jan 2006.

[63] L. Zamstein, A. Arroyo and S. Keen, "Koolio: Path planning using reinforcement learning on real robot platform,"in Conference on Recent Advances in Robotics, 2006, pp. 1-4.

[64] S. Srilakshmi, "Path Planning for mobile robots," R&D thesis, Australia National University,Canberra, 2012.

[65] C. Szepesvari, "The Asymptotic Convergence-Rate of Q-learning," Research Group on Artificial intelligence,Jozsef Attila" University, Badapest, 1998.

[66] I. Bratko*, Programming in Prolog for Artificial intelligence. 3rd Edition*. Addison-Wesley, 2001.

[67] A. Aho, J. Hopcroft and J. Ullman, *Data Structures and Algorithms*,New Jersey: Addison-Wesley, 1983.

[68] Q. Zhang, M. Li and X. Wang, "Reinforcement Learning in Robot Path Optimization," Journal of Software, vol. 7, no. 3, pp. 657-662, 2012.

[69] B. Given and R. Parr, *An Introduction to Markov Decision Processes*, 2001 [online] Available: http://www.cs.rice.edu/$\sim$vardi/dag01/givan1.pdf

[70] E. Even-Dar and Y. Mansour, "Learning Rates for Q-learning," *Journal of Machine Learning* Research, vol. 5, pp. 1-25, 2003.

[71] A. Greenwald. CS141. Class Lecture, Topic: “Reinforcement Learning.” Brown University, 2009.

[72] S. Dini and M. Serrano, “Combining Q-Learning with Artificial Neural Networks in an Adaptive *Light Seeking Robot,”* Swarthmore College, Swarthmore, Pennsylvania, May. 6, 2012.

[73] T. Khot, “Reinforcement Learning using ANN and FOIL” Final Project, University of Wisconsin, Madison, Wisconsin, unpublished.

[74] C. Liou, Q-learning with Look-up Table and Recurrent Neural Networks as a Controller for the Inverted Pendulum Problem, Unpublished, Information Engineering Department, National Taiwan University.

[76] P. Hart, N. J. Nilsson and B. Raphael. A formal basis for the heuristic determination of minimum cost paths, IEEE Transaction on System Sciences and Cybernetics, Vol. 4, pp. 100-107, 1968.

[77] H. V. Hasselt. "Double Q-learning", in: Proceeding of the Neural Information Processing Systems, Vancouver, B.C., Canada, 2010, pp. 2613-2621.

[78] K. Patnaik and S. Anwar**,** Q Learning in Context of Approximation Spaces, Contemporary Engineering Sciences, Vol. 1, no. 1, 2008, pp. 41 – 49.

[79] M. Azar, R. Munos, M. Ghavamzadeh and H. Kappen, Reinforcement Learning with a Near Optimal Rate of Convergence, Journal of Machine Learning Research, Nov. 29, 2011.

[80] M. G. Azar, R. Munos, M. Ghavamzadeh, and B. Kappen, "Speedy q-learning," Proc. of Twenty-fifth Annual Conference on Neural Information Processing Systems (NIPS 2011), pp. 2411-2419, Dec. 2011.

[81] S. P. Boyd and C. H. Barrett, Linear *Controller Design-Limits of Performance*, Prentice-Hall, 1991, pp. 54-58.

[82] TurtleBot. “Hardware Specs,” www.willowgarage.com, [Online] Available: https://www.willowgarage.com/turtlebot/specs.

[83] P. Premakumar, “How Are Robots Used in Industry?” Internet: http://www.mathworks.com/matlabcentral/fileexchange/26248-a---a-star--search-for-path-planning-tutorial, Jan. 2, 2010, [Jan. 5, 2010].

[84] Y. Mei , Y. Lu , Y. Hu and C. Lee "Deployment of mobile robots with energy and timing constraints", *IEEE Trans. Robot.*, vol. 22, no. 3, pp.507 -522 2006.

[85] B. C. Kuo and J. Tal, editors. *DC Motors and Control Systems*, SRL Publishing Company, 1978.

[86] K. Goris, Autonomous mobile robot mechanical design. PhD thesis. 2005. [Online] Available: mech.vub.ac.be/multibody/final_works/ ThesisKristofGoris.pdf.

[87] CCG research group, Goldsmith, University of London, internet: http://ccg.doc.gold.ac.uk/teaching/artificial_intelligence/lecture3.html

[88] S. Sim, K. Ong and G. Seet, "A Foundation for Robot Learning", *The Fourth International Conference on Control and Automation*, Montreal, Canada 10-12, June. 2003, pp. 649-653.

[89] C. J. Lin and C. T. Lin, "Reinforcement learning for ART-based fuzzy adaptive learning control networks", *IEEE Trans. Neural Networks*, vol. 7, pp.709 -731 1996.

[90] M.T. Mason and K.M. Lynch. Dynamic manipulation, Proc. Int. Conf. Intelligent Robots and Systems, pages 152-159, 1993.

[91] R. Calvo, J. de Oliveira, M. Figueiredo, and R. Romero. ”A Distributed Bio-Inspired Coordination Strategy for Multiple Agent Systems Applied to Surveillance Tasks in Unknown Environments”, *International Joint Conference on Neural Networks*, San Jose, CA, USA, pp. 3248-3255, 2011.

[92] E. Uchibe, M. Asada. and K. Hosoda, Behavior coordination for a mobile robot using modular reinforcement learning. *In International Conference on Intelligent Robots and Systems*, pp. 1329-1336, 1996.

[93] S. Goschin, E. Franti, M. Dascalu and S. Osiceanu., "Combine and Compare Evolutionary Robotics and Reinforcement Learning as Methods of Designing Autonomous Robots, IEEE Congress on Evolutionary Computation, pp. 1511-1516, 2007.

[94] J. Peng and R. J. Williams, "Efficient learning and planning within the Dyna framework", *Adaptive Behavior*, vol. 1, pp. 437-454, 1993.

[95] S. Gadanho, "Reinforcement Learning in Autonomous Robots: An Empirical Investigation of the Role of Emotions." Edinburgh: PhD Thesis, University of Edinburgh, 1999.

[96] T. Yamaguchi, M. Masubuchi, K. Fujihara, and M. Yachida, "Realtime reinforcement learning for a real robot in the real environment", *Proc. IEEE Int. Conf. Intelligent Robots Syst*, pp.1321-1328, 1996.

[97] S. Enokida, T. Ohashi, T. Yoshida, T. Ejima, "Stochastic field model for autonomous robot learning", *Proceedings of the IEEE International Conference on Systems*, Man, and Cybernetics 1999, IEEE SMC '99, Volume: 2, 12-15 Oct. 1999, Pages: 752 - 757.

[98] T. Yamaguchi, M. Masubuchi, K. Fujihara, and M. Yachida, "Realtime reinforcement learning for a real robot in the real environment", *Proc. IEEE Int. Conf. Intelligent Robots Syst.*, pp.1321 -1328 1996.

[99] Lin, L., *Reinforcement Learning for Robots using Neural Networks,* Dissertation, School of Computer Science, Carnegie Mellon University. Pittsburgh, pp. 2-36, 1993.

[100] P. Werbos, P, "New Forms of Reinforcement learning :Applications and Brain-like Capabilities." Washington : National Science Foundation, Proceedings of the 15th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 1993. pp. 283 - 284.

[101] C. Raju, Y. Narahari and K. Ravikumar, "Reinforcement Learning Applications in Dynamic Pricing of Retail Markets." Bangalore : Indian Institute of Science, 2003. Proceedings of the IEEE International Conference on E-Commerce. pp. 339 - 346.

[102] D. Poole and A. Mackworth, *Artificial Intelligence: Foundations of Computational Agents,* Cambridge University Press, 2010. [On-line]. Available: http://artint.info/html/ArtInt.html.

[103] Robin, "**A STAR ALGORITHM**," intelligent.worldof computing.net, Dec. 2009 [Online]. Available: http://intelligence.worldofcomputing.net/ai-search/a-star-algorithm.html. [Dec. 18 2009].

10 APPENDICES

10.1 Q-learning algorithm validation motion plots

5×5 Environments

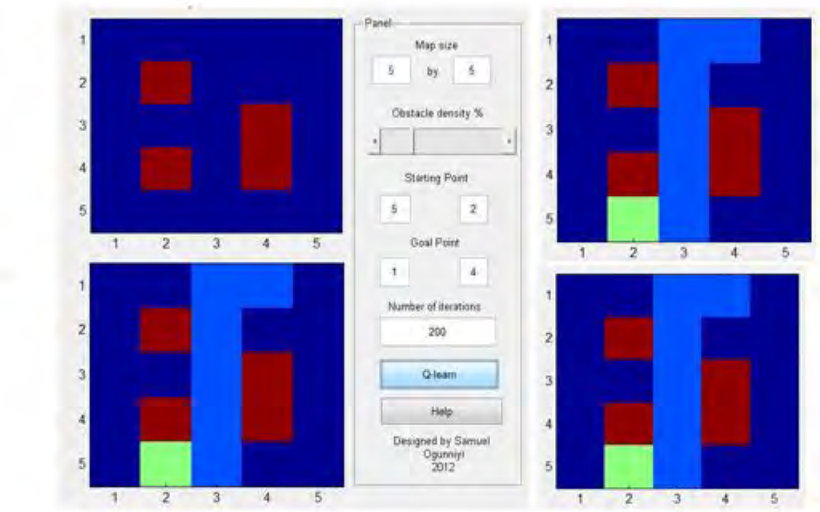

Figure 10.1: Environment 1 with start coordinates (5, 2) and target coordinates (1, 4)

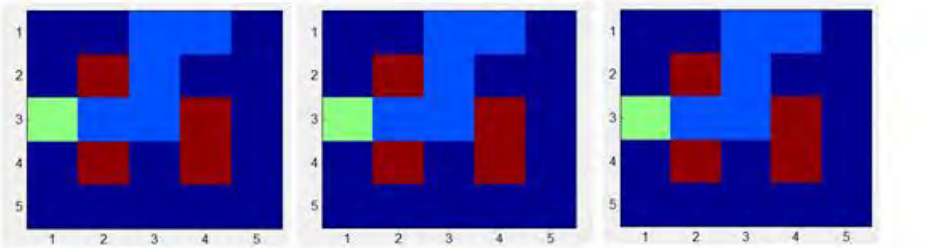

Figure 10.2: Environment 1 with start coordinates (3, 1) and target coordinates (1, 4)

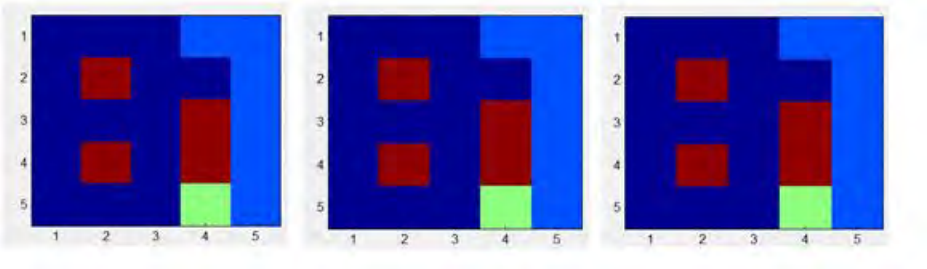

Figure 10.3: Environment 1 with start coordinates (5, 4) and target coordinates (1, 4)

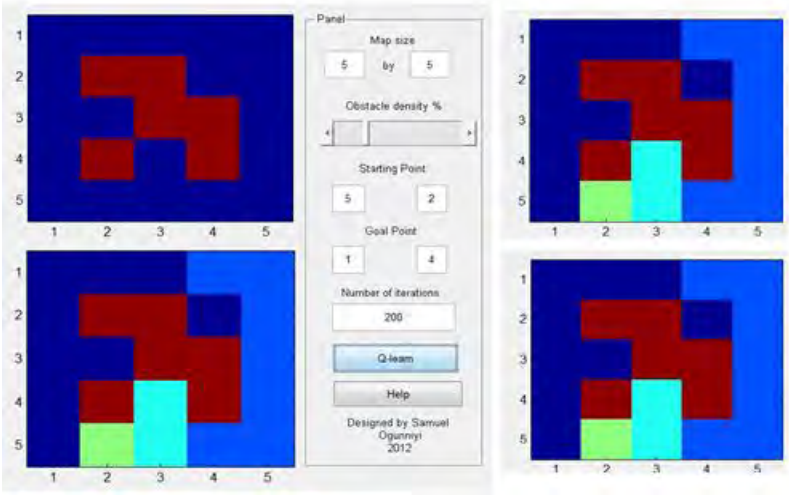

Figure 10.4: Environment 2 with start coordinates (3, 1) and target coordinates (1, 4)

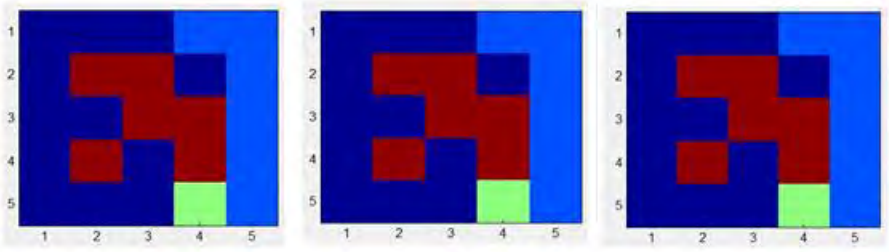

Figure 10.5: Environment 2 with start coordinates (5, 4) and target coordinates (1, 4)

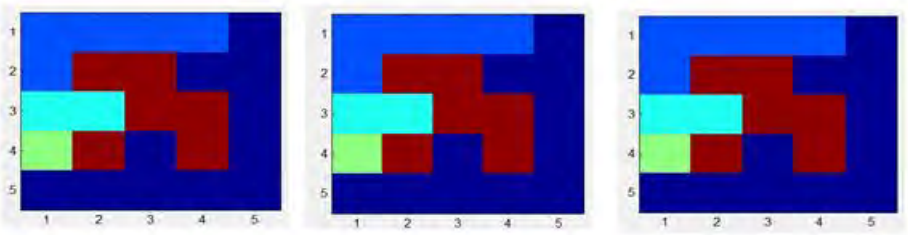

Figure 10.6: Environment 2 with start coordinates (4, 1) and target coordinates (1, 4)

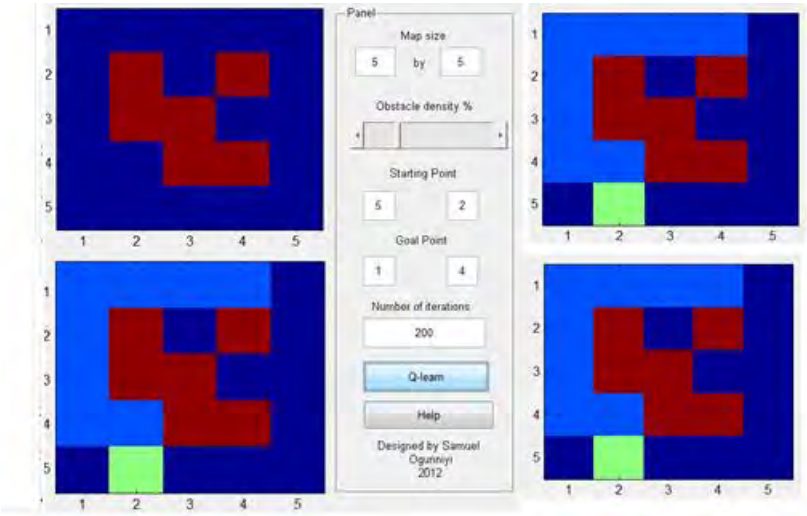

Figure 10.7: Environment 2 with start coordinates (5, 2) and target coordinates (1, 4)

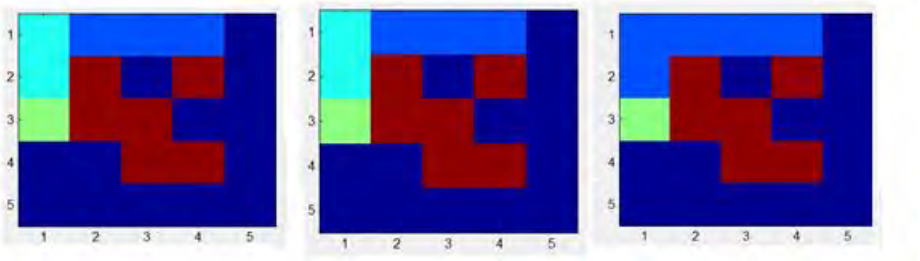

Figure 10.8: Environment 3 with start coordinates (3, 1) and target coordinates (1, 4)

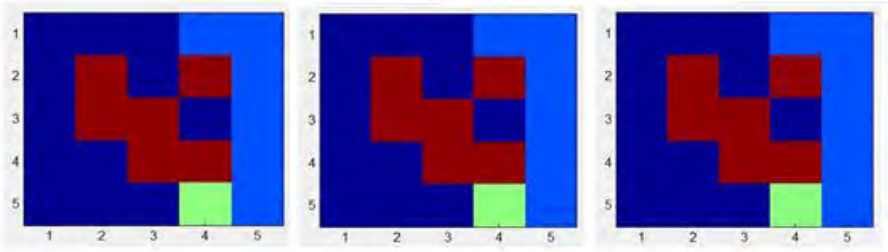

Figure 10.9: Environment 3 with start coordinates (5, 4) and target coordinates (1, 4)

10×10 Environments

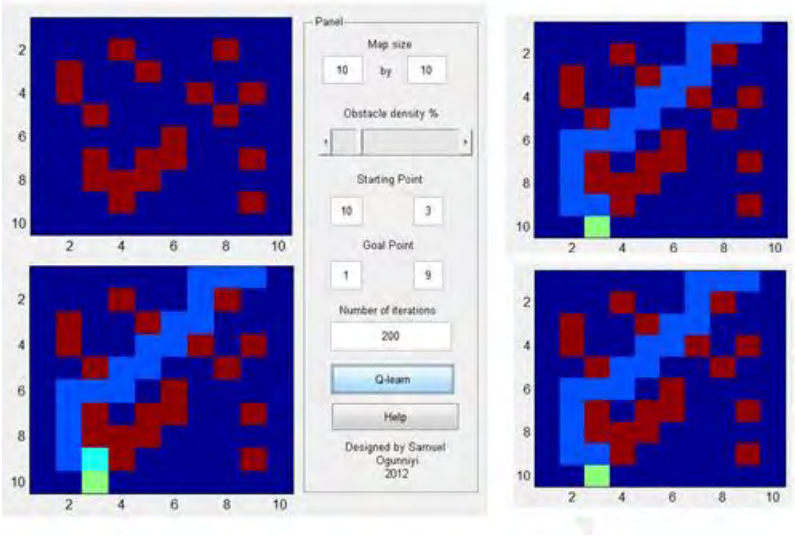

Figure 10.10: Environment 1 with start coordinates (10, 2) and target coordinates (1, 9)

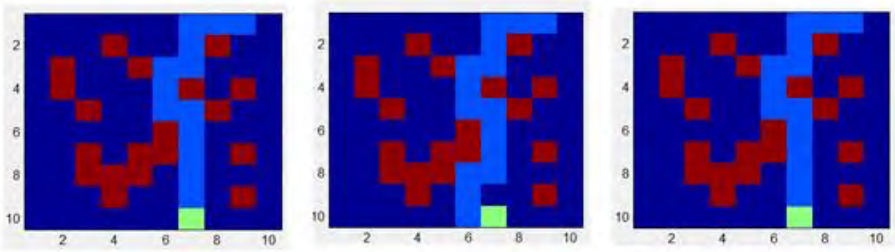

Figure 10.11: Environment 1 with start coordinates (10, 7) and target coordinates (1, 9)

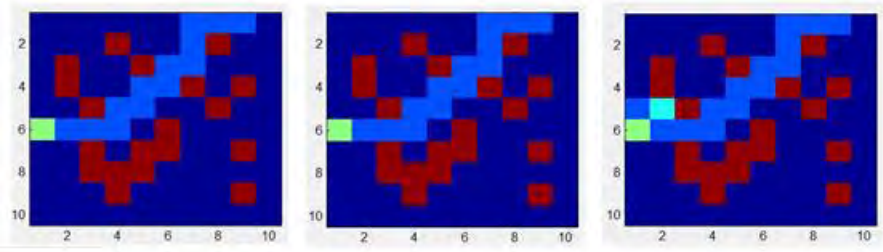

Figure 10.12: Environment 1 with start coordinates(6, 1) and target coordinates(1, 9)

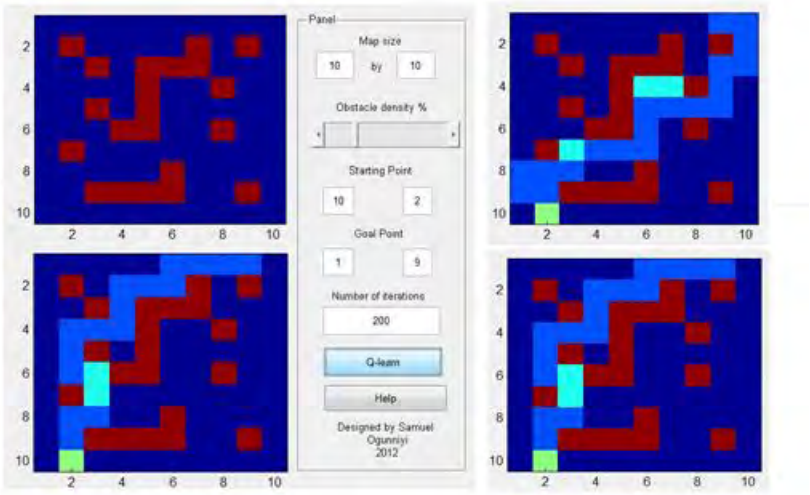

Figure 10.13: Environment 2 with start coordinates(10, 2) and target coordinates(1, 9)

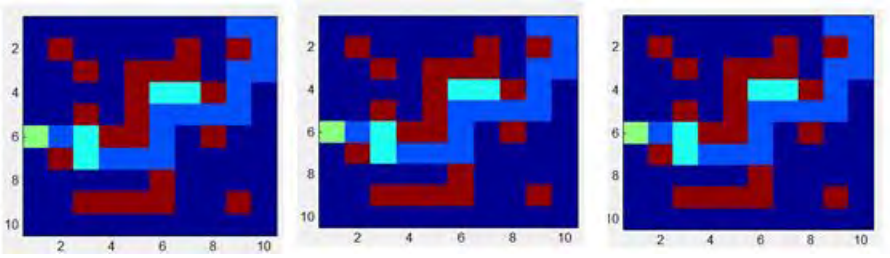

Figure 10.14: Environment 2 with start coordinates(6, 1) and target coordinates(1, 9)

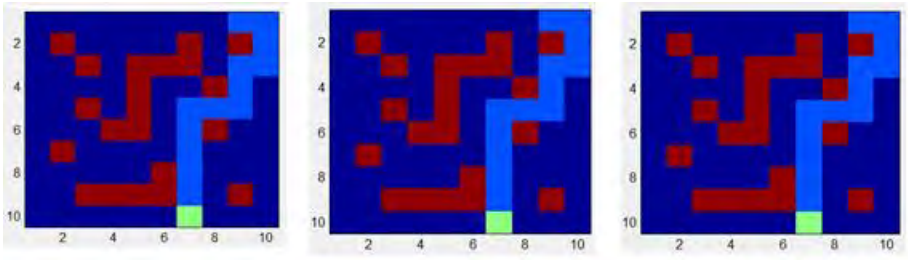

Figure 10.15: Environment 2 with start coordinates(10, 7) and target coordinates(1, 9)

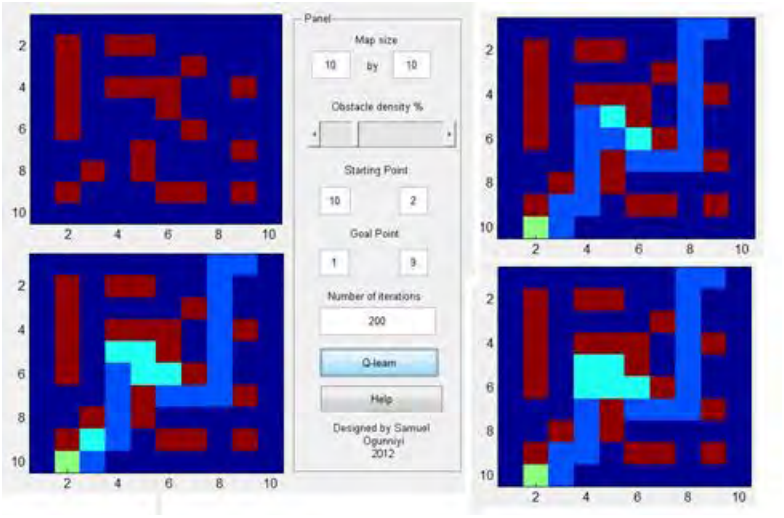

Figure 10.16: Environment 3 with start coordinates(10, 2) and target coordinates(1, 9)

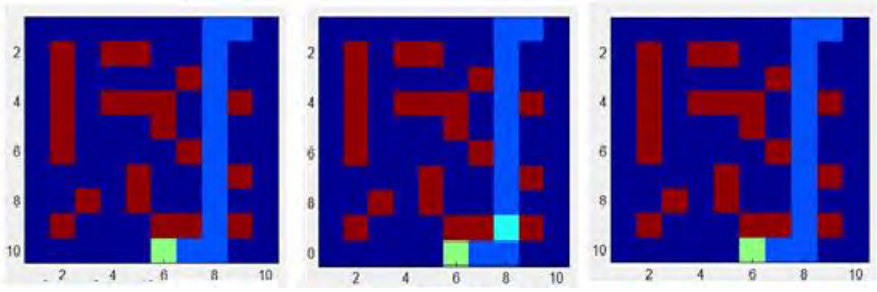

Figure 10.17: Environment 3 with start coordinates(10, 6) and target coordinates(1, 9)

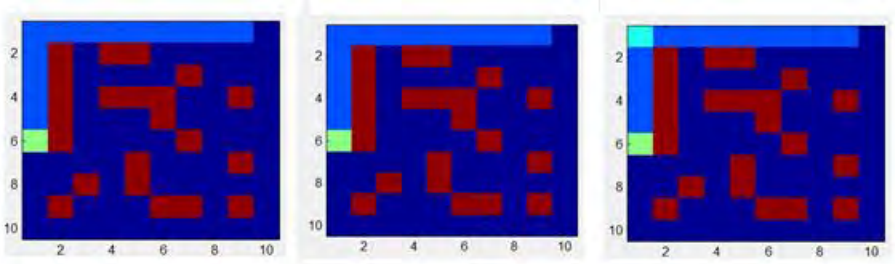

Figure 10.18: Environment 3 with start coordinates(6, 1) and target coordinates(1. 9)

Environments

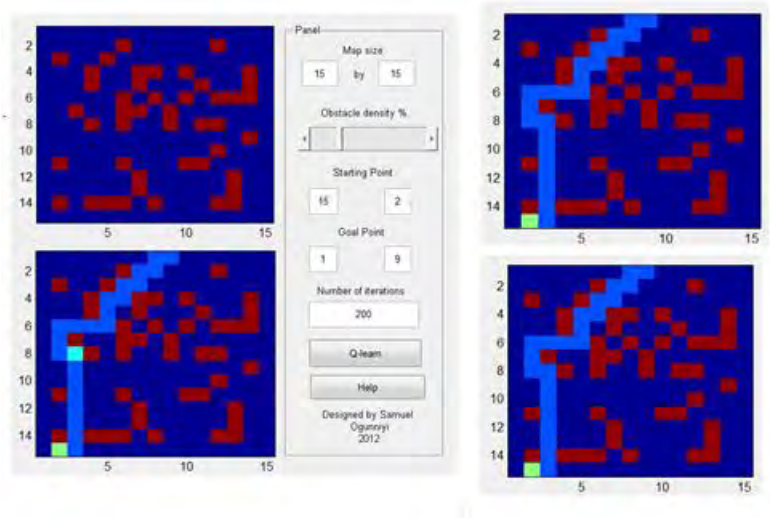

Figure 10.19: Environment 1 with start coordinates(15, 2) and target coordinates(1, 9)

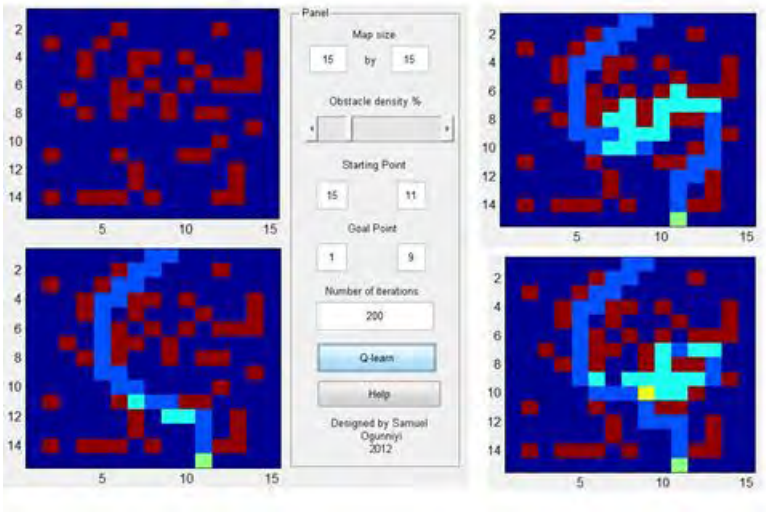

Figure 10.20: Environment 1 with start coordinates(15, 1 1) and target coordinates(1, 9)

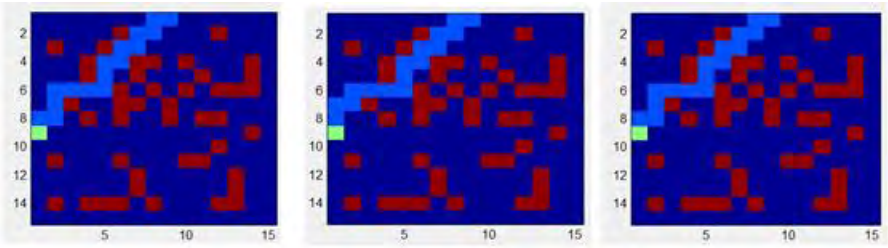

Figure 10.21: Environment 1 with start coordinates(9, 1) and target coordinates(1, 9)

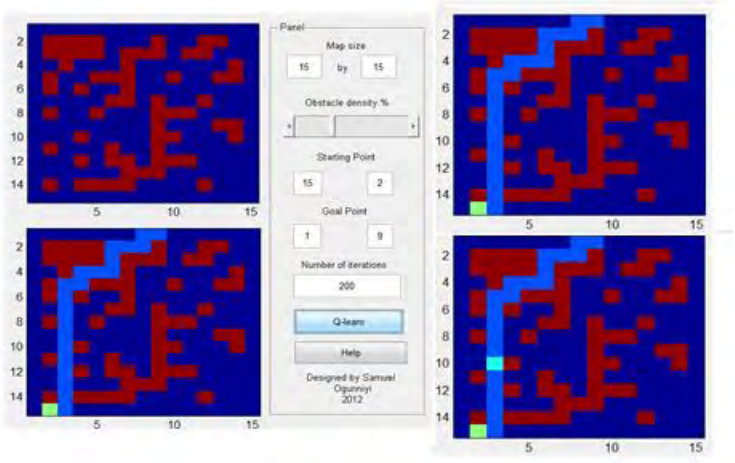

Figure 10.22: Environment 2 with start coordinates(15, 2) and target coordinates(1, 9)

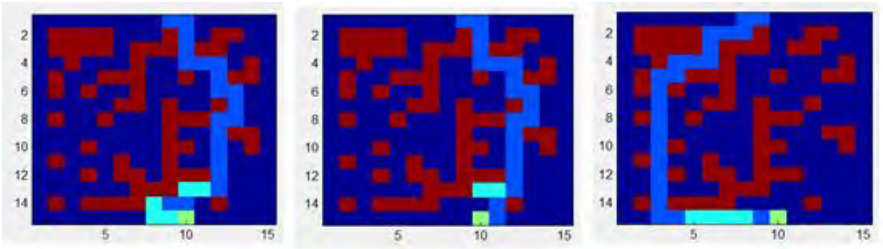

Figure 10.23: Environment 2 with start coordinates(15, 10) and target coordinates(1, 9)

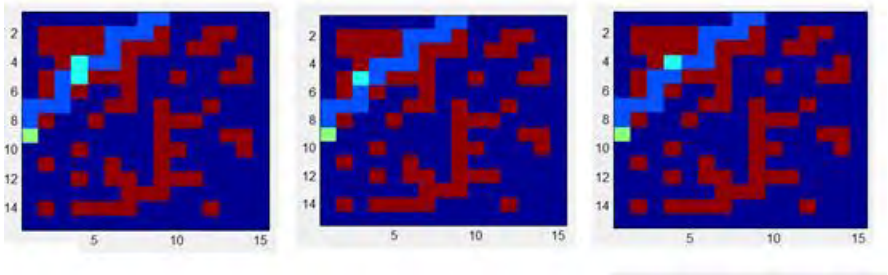

Figure 10.24: Environment 2 with start coordinates(9, 1) and target coordinates(1, 9)

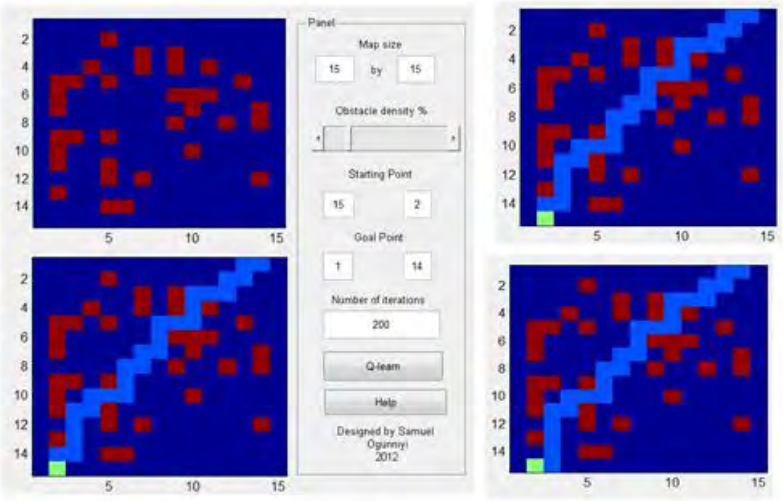

Figure 10.26: Environment 3 with start coordinates(15, 2) and target coordinates(1, 14)

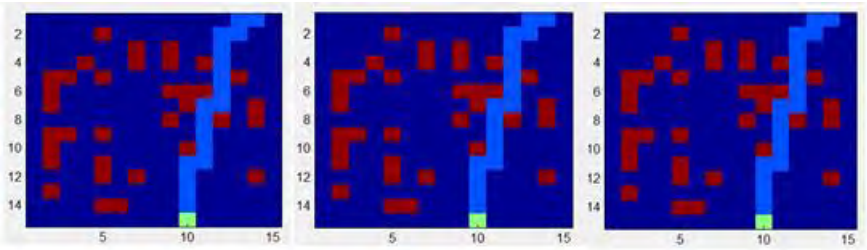

Figure 10.27: Environment 3 with start coordinates(15, 10) and target coordinates(1, 14)

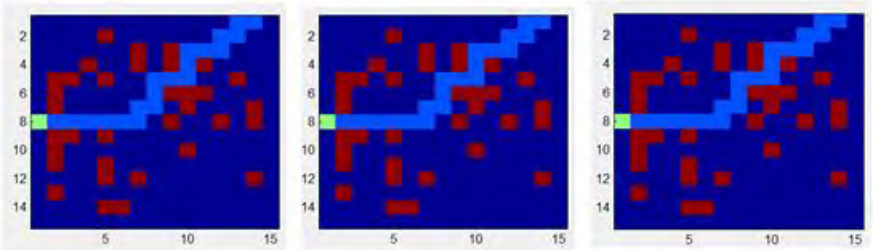

Figure 10.28: Environment 3 with start coordinates(8, 1) and target coordinates(1, 14)

20×20 Environments

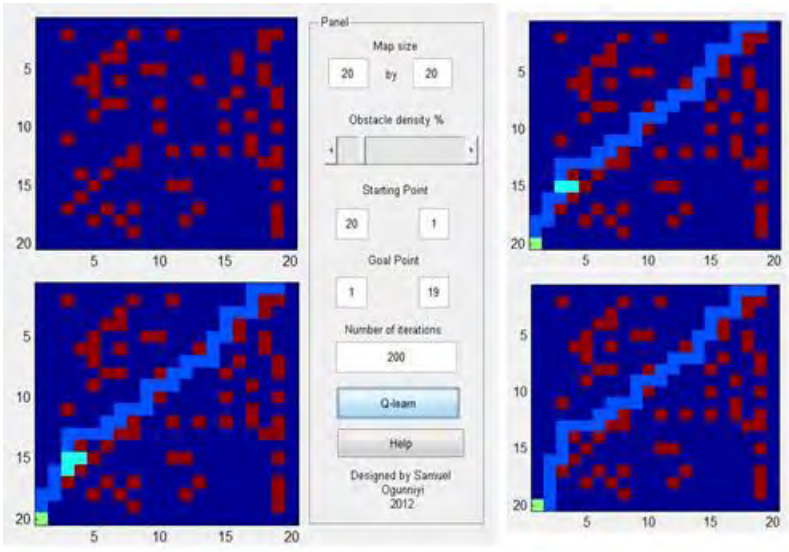

Figure 10.29: Environment 1 with start coordinates(20, 1) and target coordinates(1, 19)

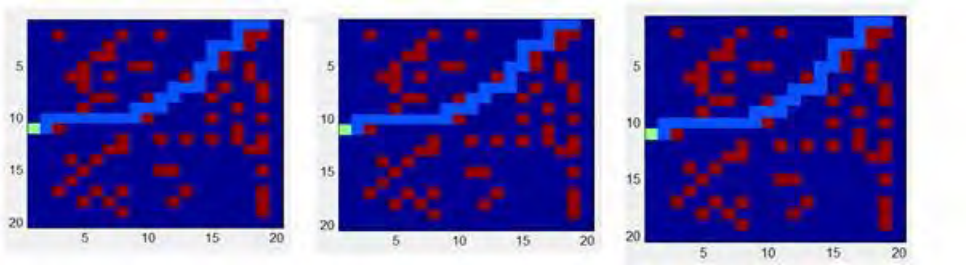

Figure 10.30: Environment 1 with start coordinates(11, 1) and target coordinates(1, 19)

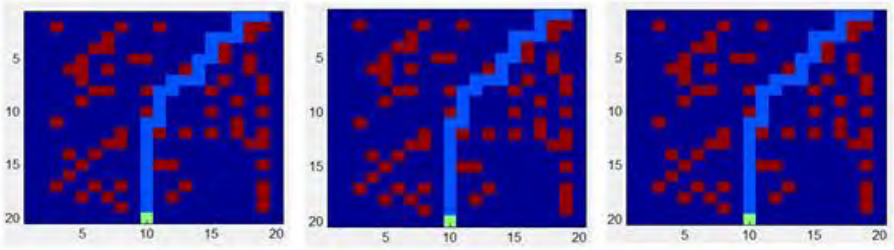

Figure 10.31: Environment 1 with start coordinates(20, 10) and target coordinates(1, 19)

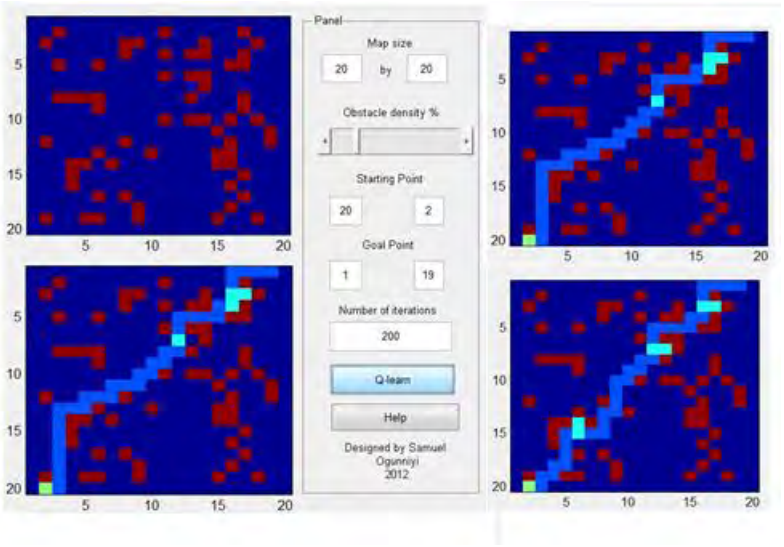

Figure 10.32: Environment 2 with start coordinates(20, 1) and target coordinates(1, 19)

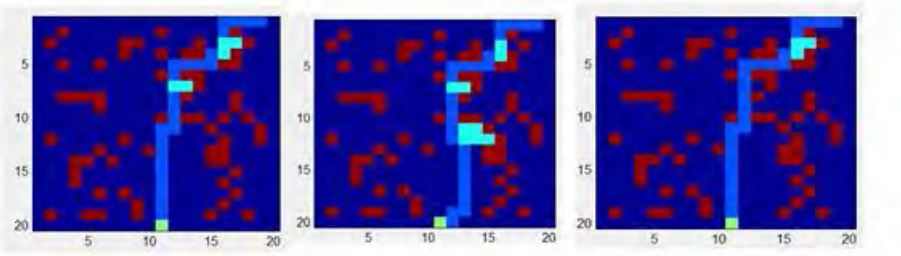

Figure 10.33: Environment 2 with start coordinates(11, 1) and target coordinates(1, 19)

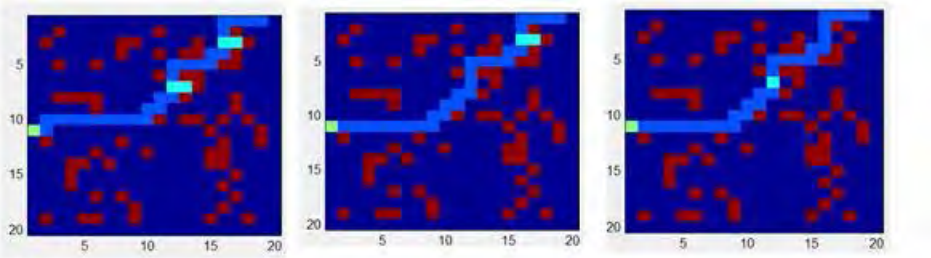

Figure 10.34: Environment 2 with start coordinates(11, 1) and target coordinates(1, 19)

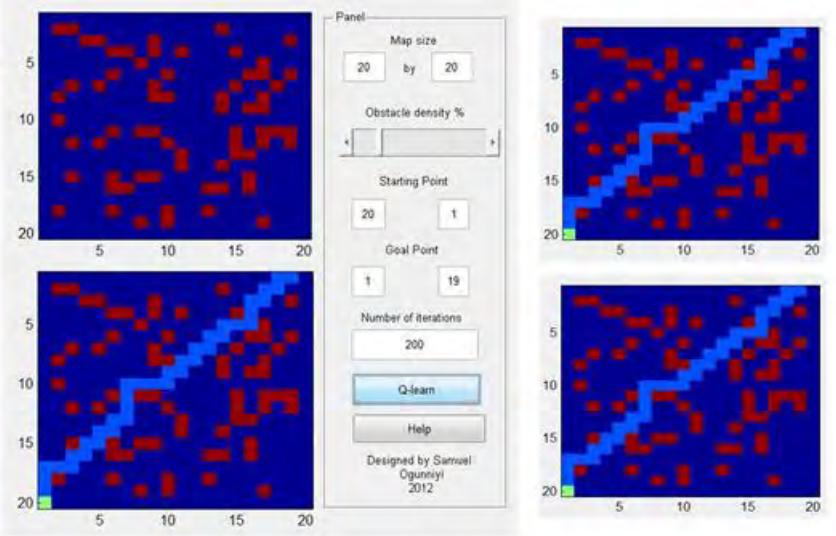

Figure 10.35: Environment 3 with start coordinates(20, 1) and target coordinates(1, 19)

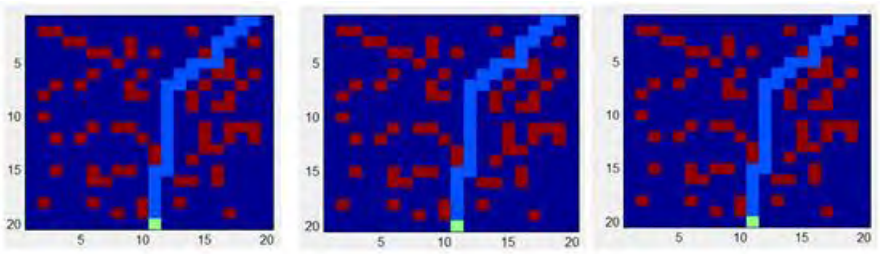

Figure 10.36: Environment 3 with start coordinates(20, 11) and target coordinates(1, 19)

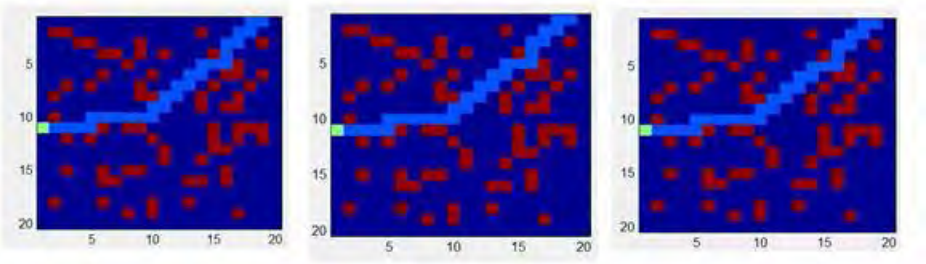

Figure 10.37: Environment 3 with start coordinates(8, 1) and target coordinates(1, 19)

10.2 Q-learning path planner code

```
% --- QLEARNER, by itself, creates a new QLEARNER object
function [varargout] = qlearner(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @qlearner_OpeningFcn, ...
                   'gui_OutputFcn',  @qlearner_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State,
varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code



% --- Executes as qlearner is made visible.
function qlearner_OpeningFcn(hObject, eventdata, handles,
varargin)

guidata(hObject, handles);
clc

global X Y Q; % Initialize the variable of the map dimension
% and the Q-table as global variables

X = str2double(get(handles.map_x_dim,'String'));
% Event listener that allows user to enter X dimension of
% grid map
Y = str2double(get(handles.map_y_dim,'String'));
% Event listener %that allows user to enter Y dimension of
% grid map

m = zeros(X, Y);% Initializes empty grid map
Q = zeros(X*Y, 8);%Initializes an empty Q-table of the size
% of memory required by the multiplication of the dimensions
% of the map and the 8 motion directions
imagesc(m)% Displays empty grid map of dimensions X and Y

end



% --- definitions of output parameters of qlearner function
function varargout = qlearner_OutputFcn(hObject, eventdata,
handles)
global X Y MAP STATE_MAP PATH OB;% Sets the dimension,map,path
% and obstacle density variables to global

varargout = {X Y MAP STATE_MAP PATH OB};
% varargout{1} = handles.output;
end

end
```

```
q_previous = Q;

axes(handles.path_disp); % Switces focus to this axes object.

 for i = 1:1% number of simulation runs

    PATH =
qlearn(start_row,start_col,goal_row,goal_col,num_iter,'q');%call
s on the path planner function to obtain planned path

    imagesc(PATH); %Displays planned path on grid map


    ENERGY_CONSUMPTION(i)= ENERGY_LOSS; %saves output energy
loss for each simulation run in an energy consumption vector
    MOTION(i) = STEPS; %Saves number of steps per simulation run
in a motion vector

    ENERGY_LOSS = 0; %Sets energy loss and steps to zero
    STEPS = 0;

    SIMULATIONS_COMPLETED = i
    i = i + 1; %iterates though the energy consumption, and
motion vectors in order to store each new simulation values in
the next elements  in the vectors.

    AVERAGE_ENERGY_LOSS = mean(ENERGY_CONSUMPTION) %computes the
average of the energy consumption vector
    AVERAGE_STEPS = mean(MOTION) %computes the average of the
motion vector

 end

end
 % Event listener that allows user to enter goal row value on
 % the grid map
 goal_col = str2double(get(handles.goal_column_field,'String'));
 % Event listener that allows user to enter goal column value on
 % the grid map

 num_iter = str2double(get(handles.iterations_field,'String'));
 % Event listener that allows user to enter the number of
 % iterations of the algorithm



 function map_x_dim_Callback(hObject, eventdata, handles)
 %function allows for the X dimension of the grid map to be
 changed at any time after the gui is opened global X Y MAP Q;
 %Sets dimension, map and Q-table matrix to global from within
 the X dimension function

 X = str2double(get(handles.map_x_dim,'String')); % Event
 listener that allows user to enter X dimension of grid map
 obst = get(hObject,'Value')/get(hObject,'Max');% allows obstacle
 density to be changed
 Q = zeros(X*Y,8); %Initializes an empty Q-table of the size of
 memory required by the multiplication of the dimensions of the
 map and the 8 motion directions

 [MAP,~] = Obstacles(obst,X,Y); %Calls the obstacle function to
 set new obstacle density
 axes(handles.map_disp); % Switches focus to this axes object.
 imagesc(MAP); % displays grid map with new obstacle density

 end
```

```
function map_y_dim_Callback(hObject, eventdata, handles)
% function allows for the Y dimension of the grid map to be
% changed at any time after the gui is opened

global X Y MAP Q; % Sets dimension, map and Q-table matrix to
% global from within the Y dimension function

Y = str2double(get(handles.map_y_dim,'String'));
% Event listener that allows user to enter Y dimension of
% grid map
obst = get(hObject,'Value')/get(hObject,'Max');
% allows obstacle density to be changed
Q = zeros(X*Y,8); % Initializes an empty Q-table of the size of
% memory required by the multiplication of the dimensions of
% the map and the 8 motion directions

[MAP,~] = Obstacles(obst,X,Y); % Sets dimension, map and Q-table
% matrix to global from within the X dimension function
axes(handles.map_disp); % Switches focus to this axes object.
imagesc(MAP); % displays grid map with new obstacle density

end

% --- Obstacle density function
function [map,n_obst] = Obstacles(obst,x,y)
% creates obstacles for a given map size
z = x*y;
map = zeros(x,y);
n_obst = ceil(obst*z);

% Creating Obstacles
for i = 1:n_obst
    ob_x = floor(2+(x-2).*rand);  % setting obstacles
    ob_y = floor(2+(y-2).*rand);
    map(ob_x,ob_y)= 10; % changing value on the grid map as to
% position obstacles
end

imagesc(map)% display grid map
map

end
```

```
% --- function for q-learner path planner
function path = qlearn(start_row,start_col,goal_row,goal_col,num_iter,~)

global MAP X Y STATE_MAP Q ENERGY_LOSS STEPS; % Sets map  dimensions,state
map, Q-table matrix ,
% energy loss and steps variable to global from within the q-learner function

Z=X*Y;
if nargin<10
    % intialising empty Q table, reward and counter matrices
    Q = zeros(Z,8);
end

% Initialising necessary vectors,
state_counter = 1;
reward_v =  zeros(1,Z);
% Intialising 1 by 2 vectors which coordinates relate to the x and y  %
coordinates on the map
goal = [goal_row,goal_col];
target = STATE_MAP(goal(1),goal(2));
start = [start_row,start_col];

MAP(goal(1),goal(1))=0; % ensure goal is not occupied
path = MAP;  % copy map to matrix called path

mystack = {}; %Initialising stack

% Setting target reward
Q(target,:)= Z;
reward_v(target) = Z;
% This where the learning rate and discount factor  is set.
alpha = 1;
gamma = 0.9;
action = 0;
% These variablse are used to  allow easier navigation through the states
iter = 1;
prev_action = 0;
ENERGY_LOSS = 0;
E1 = 0;


fprintf('Exploring...\n') % status bar
for i = 1:num_iter % Loops through number of iterations

% Progress Bar
if(mod(i,5)==1)
    UpdateProgress(100*i/num_iter);% displays the progress of the simulation
end

current_loc = [ceil(rand*X),ceil(rand*Y)]; % Ensures a random starting point
% for each simulation run which enables more of the map to be searched
% during the exploration phase

% Makes a copy of the grid map upon which the movement of the robot will be
% displayed
explore_path = MAP;
```

```
    step_counter = 0;
    % While we havent reached the goal
        while sum(current_loc ~= goal)
            % Setting possible motions
            down = [1, 0]; % South
            up = [-1, 0];  % North
            right = [0,1]; % East
            left = [0,-1]; % West

            ne = [-1,1]; % North East
            nw = [-1,-1];% North West
            se = [1,1];  % South East
            sw = [1,-1]; % South West

            %  Checks the distances to centre of all 8 surrounding states

            Ed_d = sqrt((goal-(current_loc+down))*(goal-(current_loc+down))');
            Ed_u = sqrt((goal-(current_loc+up))*(goal-(current_loc+up))');
            Ed_r = sqrt((goal-(current_loc+right))*(goal-(current_loc+right))');
            Ed_l = sqrt((goal-(current_loc+left))*(goal-(current_loc+left))');

            Ed_ne = sqrt((goal-(current_loc+ne))*(goal-(current_loc+ne))');
            Ed_nw = sqrt((goal-(current_loc+nw))*(goal-(current_loc+nw))');
            Ed_se = sqrt((goal-(current_loc+se))*(goal-(current_loc+se))');
            Ed_sw = sqrt((goal-(current_loc+sw))*(goal-(current_loc+sw))');

            Ed = [Ed_d Ed_u Ed_r Ed_l,Ed_ne,Ed_nw,Ed_se,Ed_sw];

            % Calculates the possible next states by adding the vectors of the 8
            % possible motion to the current state vector
            possible_next_locations =[(current_loc+down);(current_loc+up);...
            (current_loc+right);(current_loc+left);...
            (current_loc+ne);(current_loc+nw);(current_loc+se);(current_loc+sw)];

            loop_counter=0;
            Egreedy = rand(1);



        % --- Egreedy action selection policy
         if Egreedy < 0.8
            p_action = action;
            [minDist,action] = min(Ed); % calculates which surrounding state
% has the minimum
            euclidean distance from the current state
            next_loc = possible_next_locations(action,:);
            % selecting a valid random action in current state (no obstacles)
            while sum(next_loc<1) || next_loc(1)>X || next_loc(2)>Y ||
            % While loop ensures that the next location does not lie on a
 % boundary
            explore_path(next_loc(1),next_loc(2)) > 2
            Ed(action) = minDist*2;
           [minDist,action] = min(Ed);

            next_loc = possible_next_locations(action,:);
            if loop_counter>30
                % eliminating the possibility of choosing impossible
```

```
        % movements i.e. into the border of the map
                        if (current_loc(1)+ down(1))>X ||...
                            explore_path(current_loc(1)+1,current_loc(2)) > 9
                            down = [0, 0];
                        end
                        if (current_loc(1)+up(1))<1 ||...
                            explore_path(current_loc(1)-1,current_loc(2)) > 9
                            up = [0, 0];
                        end
                        if (current_loc(2)+right(2))>Y ||...
                            explore_path(current_loc(1),current_loc(2)+1) > 9
                            right = [0, 0];
                        end
                        if (current_loc(2)+left(2))<1 ||...
                            explore_path(current_loc(1),current_loc(2)-1) > 9
                            left = [0, 0];
                        end

                        if (current_loc(1)+ne(1))< 1 ||...
                       (current_loc(2)+ne(2))> Y || explore_path(current_loc(1)-1,...
                           current_loc(2)+ 1) > 9
                          ne = [0,0];
                        end
                        if (current_loc(1)+ nw(1))< 1 ||...
                       (current_loc(2)+nw(2))< 1 || explore_path(current_loc(1)-1,...
                             current_loc(2)-1) > 9
                            nw = [0,0];
                        end
                        if (current_loc(1)+se(1))> X ||...
                           (current_loc(2)+se(2))> Y ||
        explore_path(current_loc(1)+1,...
                            current_loc(2)+1) > 9
                        se = [0,0];
                        end
                        if (current_loc(1)+ sw(1))> X||...
                           (current_loc(2)+sw(2))< 1 ||
        explore_path(current_loc(1)+1,...
                            current_loc(2)-1) > 9
                            sw = [0, 0];
                        end



      % Calculates the possible next states by adding the vectors of the 8
      % possible motion to the current state vector
possible_next_locations = [(current_loc+down); (current_loc+up);...
    (current_loc+right); (current_loc+left);...
(current_loc+ne);(current_loc+nw);(current_loc+se);(current_loc+sw) ];

                action = p_action;
                next_loc = possible_next_locations(action,:);

                break
                end
                loop_counter = loop_counter+1;

            end
        else
```

```
            action = round((7*rand)+1); % this is the random action selector
% section of the e-greedy policy
            next_loc = possible_next_locations(action,:);

        while sum(next_loc<1) || next_loc(1)>X || next_loc(2)>Y ||
explore_path(next_loc(1),next_loc(2)) > 3
            action = round((7*rand)+1);
            next_loc = possible_next_locations(action,:);
        if loop_counter>30
            % eliminating the possibility of choosing impossible movements
% i.e. into the border of the map
              if (current_loc(1)+down(1))>X ||...
                  explore_path(current_loc(1)+1,current_loc(2)) > 9
                  down = [0, 0];
              end
              if (current_loc(1)+up(1))<1 ||...
                  explore_path(current_loc(1)-1,current_loc(2)) > 9
                  up = [0, 0];
              end
              if (current_loc(2)+right(2))>Y ||...
                  explore_path(current_loc(1),current_loc(2)+1) > 9
                  right = [0, 0];
              end
              if (current_loc(2)+left(2))<1 ||...
                  explore_path(current_loc(1),current_loc(2)-1) > 9
                  left = [0, 0];
              end
              if (current_loc(1)+ne(1))< 1 ||...
                 (current_loc(2)+ne(2))> Y ||...
                explore_path(current_loc(1)-1,current_loc(2)+ 1) > 9
                  ne = [0,0];
              end
              if (current_loc(1)+ nw(1))< 1 ||...
             (current_loc(2) +nw(2))< 1  || explore_path(current_loc(1)-1,...
                  current_loc(2)-1) > 9
                   nw = [0,0];
              end
              if (current_loc(1)+se(1))> X ||...
                 (current_loc(2)+se(2))>Y   ||
explore_path(current_loc(1)+1,...
                  current_loc(2)+1) > 9
                  se = [0,0];
              end
              if (current_loc(1)+ sw(1))> X||...
                 (current_loc(2)+sw(2))< 1  ||
explore_path(current_loc(1)+1,...
                  current_loc(2)-1) > 9
                   sw = [0, 0];
              end


        possible_next_locations = [(current_loc+down);....
          (current_loc+up); (current_loc+right);...
        (current_loc+left);(current_loc+ne);(current_loc+nw);...
        (current_loc+se);(current_loc+sw)];

        action = round((7*rand)+1);
```

```
                next_loc = possible_next_locations(action,:);
                break
            end
            loop_counter = loop_counter+1;

        end
    end

        current_state = STATE_MAP(current_loc(1),current_loc(2));
% checks current state number
        state_counter = state_counter+1; % counts how many times the robot
% has been in current state
        alpha = 1/( state_counter);

        next_state = STATE_MAP(next_loc(1),next_loc(2));
 % checks next state number
        reward = reward_v(current_state); % checks the corresponding reward
% for moving to the current state

        % Push data onto Stack
        mystack{end+1} = [action,current_state,next_state];


        prev_loc = current_loc;
        % moves to next state
        current_loc = next_loc;

        % stores movement in explore_path grid map
        if explore_path(current_loc(1),current_loc(2))>0
            explore_path(prev_loc(1),prev_loc(2))...
           = explore_path(prev_loc(1),prev_loc(2))+2;
        end

        % Draw/Plot explore_path on map
        explore_path(current_loc(1),current_loc(2))...
        = explore_path(current_loc(1),current_loc(2))+2;

        if step_counter > Z/2 % step counter that checks if robot has been in
% one state for too long
            break
        end
        step_counter = step_counter+1; % while we havent reached the goal %
% counter number of steps taken thus fare
    end


% pops the stored state action pair information into the Q-table for the
% purpose of updating
for j = length(mystack):-1:1

    x = mystack{j};

    action = x(1);
    current_state = x(2);
    next_state = x(3);
```

```
        % Storing all the popped values into the Q-table
        Q(current_state,action)= Q(current_state,action) + ...
        alpha*(reward + gamma*max(Q(next_state))- Q(current_state,action));
        reward_v(current_state) = sum(Q(current_state,:));

    end

    end% for all iterations
    fprintf('Exploring...Done\n') %Status that notifies that exploration is done

    % Intialising variables before navigation commences
    steps = 0;
    path(start(1),start(2))= 5; % number used to distinguish starting state on
    the % path map
    current_loc = start;
    m = 1;

    fprintf('Navigating...\n')
    while sum(current_loc ~= goal)% Navigation stage
    % Navigates until goal state found

    down = [1, 0]; % South
    up = [-1, 0];  % North
    right = [0,1]; % East
    left = [0,-1]; % West
    ne = [-1,1];   % North East
    nw = [-1,-1];  % North West
    se = [1,1];    % South East
    sw = [1,-1];   % South West

    % eliminating the possibility of choosing impossible movements i.e. into the
    border of the map

    if (current_loc(1)+down(1))>X || ...
        explore_path(current_loc(1)+1,current_loc(2)) > 9
        down = [0, 0];
    end
    if (current_loc(1)+up(1))<1  ||...
        explore_path(current_loc(1)-1,current_loc(2)) > 9
        up = [0, 0];
    end
    if (current_loc(2)+right(2))>Y ||...
        explore_path(current_loc(1),current_loc(2)+1) > 9
       right = [0, 0];
    end
    if (current_loc(2)+left(2))<1 ||...
    explore_path(current_loc(1),current_loc(2)-1) > 9
    left = [0, 0];
    end


if (current_loc(1)+ne(1))< 1 || (current_loc(2)+ne(2))> Y ||...
    explore_path(current_loc(1)-1,current_loc(2)+ 1) > 9
     ne = [0,0];
end
if (current_loc(1)+ nw(1))< 1 || (current_loc(2) +nw(2))< 1 ||...
```

```
        explore_path(current_loc(1)-1,current_loc(2)-1) > 9
        nw = [0,0];
    end
    if (current_loc(1)+se(1))> X || (current_loc(2)+se(2))>Y ||...
        explore_path(current_loc(1)+1,current_loc(2)+1) > 9
        se = [0,0];
    end
    if (current_loc(1)+ sw(1))> X|| (current_loc(2)+sw(2))< 1 ||...
        explore_path(current_loc(1)+1,current_loc(2)-1) > 9
        sw = [0, 0];
    end

    possible_next_locations = [(current_loc+down); (current_loc+up);...
    (current_loc+right); (current_loc+left);(current_loc+ne);(current_loc+nw);...
    (current_loc+se);(current_loc+sw) ];

    p_action = action;
    e = STATE_MAP(current_loc(1),current_loc(2));% checks which state the robot
    % is in
    [~,action]= max(Q(e,:)); % computes the action with the maximum Q-value in
    % the current state
    q_dummy = Q; % stores the Q-table in a temporary table

    next_loc = possible_next_locations(action,:);% assigns next location based on
    % chosen action
    loop_counter = 0;

    while sum(next_loc<1) || next_loc(1)>X || next_loc(2)>Y...
            || path(next_loc(1),next_loc(2))>2
        q_dummy(e,action) = 0;% If the robot's next location is not into a
    % boundary or an obstacle
    % the Q-value of the corresponding action gets set to zero.

        [~,action]= max(q_dummy(e,:));% If the action chosen was not possible the
    % robot must chose the
        % action whose Q-value is the next biggest.

    % The loop is broken if the robot stays in one state too long, indicating it
    % is stuck.
        if loop_counter>30
            action = p_action;
            fprintf('Navigation Break')
            break
        end

        next_loc = possible_next_locations(action,:);
        loop_counter = loop_counter+1;
    end
    % This section assigns energy loss value of choosing any of the possible 8
    motions,

% while entering any of the 8 neighbour states
    if prev_action == 0
  ENERGY_LOSS = 0;
  El_d = 0;El_u = 0;El_r = 0;El_l = 0;El_ne = 0;El_nw = 0;El_se = 0;
  El_sw = 0;
```

```
    El = [El_d El_u El_r El_l,El_ne,El_nw,El_se,El_sw];
    Et = 0;
    elseif prev_action == 1
    El_d = 0;El_u = 0.38;El_r = 0.21;El_l = 0.19;El_ne = 0.29;El_nw =
0.30;El_se = 0.14;El_sw = 0.13;
    El = [El_d El_u El_r El_l,El_ne,El_nw,El_se,El_sw];
    Et = 0.34;
     elseif prev_action == 2
    El_d = 0.38;El_u = 0;El_r = 0.19;El_l = 0.21;El_ne = 0.13;El_nw =
0.14;El_se = 0.30;El_sw = 0.29;
    El = [El_d El_u El_r El_l,El_ne,El_nw,El_se,El_sw];
    Et = 0.34;
    elseif prev_action == 3
    El_d = 0.19;El_u = 0.21;El_r = 0;El_l = 0.38;El_ne = 0.14;El_nw =
0.29;El_se = 0.13;El_sw = 0.30;
    El = [El_d El_u El_r El_l,El_ne,El_nw,El_se,El_sw];
    Et = 0.34;
    elseif prev_action == 4
    El_d = 0.21;El_u = 0.19;El_r = 0.38;El_l = 0;El_ne = 0.30;El_nw =
0.13;El_se = 0.29;El_sw = 0.14 ;
    El = [El_d El_u El_r El_l,El_ne,El_nw,El_se,El_sw];
    Et = 0.34;
    elseif prev_action == 5
    El_d = 0.30;El_u = 0.14;El_r = 0.13;El_l = 0.29;El_ne = 0;El_nw =
0.21;El_se = 0.19;El_sw = 0.38 ;
    El = [El_d El_u El_r El_l,El_ne,El_nw,El_se,El_sw];
    Et = 0.4808;
    elseif prev_action == 6
    El_d = 0.29;El_u = 0.13;El_r = 0.30;El_l = 0.14;El_ne = 0.19;El_nw =
0;El_se = 0.38;El_sw = 0.21 ;
    El = [El_d El_u El_r El_l,El_ne,El_nw,El_se,El_sw];
    Et = 0.4808;
    elseif prev_action == 7
    El_d = 0.13;El_u = 0.29;El_r = 0.14;El_l = 0.30;El_ne = 0.21;El_nw =
0.38;El_se = 0;El_sw = 0.19;
    El = [El_d El_u El_r El_l,El_ne,El_nw,El_se,El_sw];
    Et = 0.4808;
    elseif prev_action == 8
    El_d = 0.13;El_u = 0.29;El_r = 0.30;El_l = 0.14;El_ne = 0.38;El_nw =
0.21;El_se = 0.19;El_sw = 0;
    El = [El_d El_u El_r El_l,El_ne,El_nw,El_se,El_sw];
    Et = 0.4808;
    end

   ENERGY_LOSS =   ENERGY_LOSS + El(action)+ Et; % Will have computed the
total energy loss when navigation is done.


prev_action = action;

prev_loc = current_loc;% moves to new state
 % MOVE
current_loc = next_loc;

% prints the robot's movements to the explore_path  map

 if explore_path(current_loc(1),current_loc(2))>0
```

```
        explore_path(prev_loc(1),prev_loc(2))...
        = explore_path(prev_loc(1),prev_loc(2))+2;
    end

% plot robot's path on path map
path(current_loc(1),current_loc(2))...
 = path(current_loc(1),current_loc(2)) + 2;

steps = steps+1;% counts number of steps taken by the robot thus far.
imagesc(path) % Displays navigated path

    pause(0.3)
end% while we havent reached the goal

ENERGY_LOSS;
STEPS = steps;
fprintf('Navigating...Done\n')% Displays that navigation is done
end
```

EBE Faculty: Assessment of Ethics in Research Projects

Any person planning to undertake research in the Faculty of Engineering and the Built Environment at the University of Cape Town is required to complete this form before collecting or analysing data. When completed it should be submitted to the supervisor (where applicable) and from there to the Head of Department. If any of the questions below have been answered YES, and the applicant is NOT a fourth year student, the Head should forward this form for approval by the Faculty EIR committee: submit to Ms Zulpha Geyer (Zulpha.Geyer@uct.ac.za; Chem Eng Building, Ph 021 650 4791).

NB: A copy of this signed form must be included with the thesis/dissertation/report when it is submitted for examination

This form must only be completed once the most recent revision EBE EiR Handbook has been read.

Name of Principal Researcher/Student: Samuel Ogunniyi Department: Electrical Engineering

Preferred email address of the applicant: samogunniyi@gmail.com

If a Student: Degree: Msc Electrical Engineering Supervisor: Mr Tsoeu

If a Research Contract indicate source of funding/sponsorship:

Research Project Title: Energy efficient path planning: *The effectiveness of a Q-learning algorithm in saving energy*

Overview of ethics issues in your research project:

Question 1: Is there a possibility that your research could cause harm to a third party (i.e. a person not involved in your project)?	YES	**NO** √
Question 2: Is your research making use of human subjects as sources of data? If your answer is YES, please complete Addendum 2.	YES	**NO** √

Question 3: Does your research involve the participation of or provision of services to communities? If your answer is YES, please complete Addendum 3.	YES	**NO**
Question 4: If your research is sponsored, is there any potential for conflicts of interest? If your answer is YES, please complete Addendum 4.	YES	**NO**

If you have answered YES to any of the above questions, please append a copy of your research proposal, as well as any interview schedules or questionnaires (Addendum 1) and please complete further addenda as appropriate. Ensure that you refer to the EiR Handbook to assist you in completing the documentation requirements for this form.

I hereby undertake to carry out my research in such a way that

- there is no apparent legal objection to the nature or the method of research; and
- the research will not compromise staff or students or the other responsibilities of the University;
- the stated objective will be achieved, and the findings will have a high degree of validity;
- limitations and alternative interpretations will be considered;
- the findings could be subject to peer review and publicly available; and
- I will comply with the conventions of copyright and avoid any practice that would constitute plagiarism.

Signed by:

	Full name and signature	Date
Principal Researcher/Student:	Samuel Ogunniyi	28/08/2014

This application is approved by:

Supervisor (if applicable):		
HOD (or delegated nominee): *Final authority for all assessments with NO to all questions and for all undergraduate research.*		
Chair : Faculty EIR Committee For applicants other than undergraduate students who have answered YES to any of the above questions.		

ADDENDUM 2: To be completed if you answered YES to Question 2:

It is assumed that you have read the UCT Code for Research involving Human Subjects (available at http://web.uct.ac.za/depts/educate/download/uctcodeforresearchinvolvinghumansubjects.pdf) in order to be able to answer the questions in this addendum.

2.1 Does the research discriminate against participation by individuals, or differentiate between participants, on the grounds of gender, race or ethnic group, age range, religion, income, handicap, illness or any similar classification?	YES	NO
2.2 Does the research require the participation of socially or physically vulnerable people (children, aged, disabled, etc) or legally restricted groups?	YES	NO
2.3 Will you not be able to secure the informed consent of all participants in the research? (In the case of children, will you not be able to obtain the consent of their guardians or parents?)	YES	NO
2.4 Will any confidential data be collected or will identifiable records of individuals be kept?	YES	NO

2.5 In reporting on this research is there any possibility that you will not be able to keep the identities of the individuals involved anonymous?	YES	NO
2.6 Are there any foreseeable risks of physical, psychological or social harm to participants that might occur in the course of the research?	YES	NO
2.7 Does the research include making payments or giving gifts to any participants?	YES	NO

If you have answered YES to any of these questions, please describe below how you plan to address these issues:

ADDENDUM 3: To be completed if you answered YES to Question 3:

3.1 Is the community expected to make decisions for, during or based on the research?	YES	NO
3.2 At the end of the research will any economic or social process be terminated or left unsupported, or equipment or facilities used in the research be recovered from the participants or community?	YES	NO
3.3 Will any service be provided at a level below the generally accepted standards?	YES	NO

If you have answered YES to any of these questions, please describe below how you plan to address these issues:

ADDENDUM 4: To be completed if you answered YES to Question 4

4.1 Is there any existing or potential conflict of interest between a research sponsor, academic supervisor, other researchers or participants?	YES	NO
4.2 Will information that reveals the identity of participants be supplied to a research sponsor, other than with the permission of the individuals?	YES	NO
4.3 Does the proposed research potentially conflict with the research of any other individual or group within the University?	YES	NO

If you have answered YES to any of these questions, please describe below how you plan to address these issues: